ENCICLOPEDIA DEL

BASIC

LeG

SPECTRUM 4

La lógica del ordenador
Gráficos y sonido



ceac

SPECTRUM 4

La lógica del ordenado
Gráficos y sonido

ENCICLOPEDIA
DEL
BASIC

SPECTRUM 4

La lógica del ordenado
Gráficos y sonido

ediciones ceac
Perú, 164 - 08020 Barcelona - España

Primera edición: Enero 1987

ISBN 84-329-7714-4 (Rústica)
ISBN 84-329-7719-5 (Rústica especial)
ISBN 84-329-7724-1 (Cartoné)
ISBN 84-329-7731-4 (Cartoné especial)
ISBN 84-329-7726-8 (Cartoné, Obra completa)
ISBN 84-329-7727-6 (Cartoné, Obra completa especial)

Depósito Legal: B-44040 - 1986

Impreso por
GERSA, Industria Gráfica
Tambor del Bruc, 6
08970 Sant Joan Despí (Barcelona)

Printed in Spain
Impreso en España

Contenido

Aclaración importante

Observará el lector de esta «Enciclopedia del BASIC» que la numeración de capítulos es correlativa, iniciándose en el Tomo I de la misma y finalizando en el Tomo V y último.

Hemos creído conveniente tal continuidad, en primer lugar por el carácter secuencial de la Enciclopedia, al ser los temas que la constituyen correlativos entre sí y formando un todo inseparable; y en segundo lugar porque estimamos que ello facilita la *rápida localización de cualquier tema,* lo que, al tratarse de una obra de consulta frecuente, supone una indudable ventaja para el lector.

Cómo estudiar en esta Enciclopedia

Al comenzar cada Capítulo encontrará un *esquema de contenido*. La finalidad de este esquema es doble:

— Por una parte, le ofrece una visión panorámica de todos los temas que va a estudiar en ese capítulo.

— Por otra, si posteriormente debe repasar algún punto determinado, le facilitará el poder localizarlo.

Leálos despacio para tener esta visión panorámica.

Después del esquema de contenido, cada capítulo comienza definiendo sus objetivos. De esta manera usted sabrá desde el principio lo que aprenderá en él. También le servirá como referencia para saber si el objetivo marcado ha sido alcanzado por usted.

A lo largo de los capítulos y al final de los mismos encontrará unos *resúmenes*, seguidos de unos *ejercicios de autocomprobación*. Su finalidad es hacer un alto en el camino y recapitular lo estudiado desde la última parada. Al mismo tiempo, los ejercicios de autocomprobación le servirán para que usted mismo compruebe si ha asimilado los conceptos estudiados y si está en disposición de continuar adelante o, por el contrario, si es necesario volver a repasar algo antes de seguir. La solución a los ejercicios de autocomprobación la encontrará al final de la primera parte del volumen.

Le recomendamos también que, cuando deba interrumpir su estudio, lo haga siempre al final de un capítulo o al terminar de resolver alguno de los grupos de ejercicios de autocomprobación que aparecen a lo largo de las lecciones.

Al hablar de la composición de la Enciclopedia, le dijimos que cada volumen se componía de dos partes. La del texto principal o estudio del BASIC estándar, y la de «Práctica con el microordenador», que viene a ser un complemento de la anterior. Ahora que va a comenzar a estudiarlo comprenderá enseguida la razón de esta comparación.

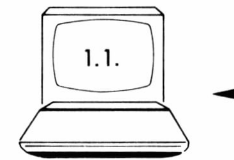
Cuando se aprende un idioma es frecuente que uno se encuentre con la sorpresa de que en determinadas regiones aquello que uno aprendió a decirlo de una manera se diga de otra. Con el lenguaje de programación BASIC pasa lo mismo. Cada fabricante introduce en el uso de su ordenador un dialecto distinto, que normalmente coincidirá en su mayor parte con el

BASIC estándar, pero tendrá suficientes características especiales como para que el que comienza se encuentre ante su ordenador sin saber qué hacer en determinados momentos.

Por eso, como nuestra intención es que aprenda BASIC y que lo practique con *su microordenador* hemos utilizado la Enciclopedia de modo que usted no se encuentre perdido en ningún momento. Así, al estudiar la lección del BASIC estándar es como si dijéramos: *Esto se dice así normalmente en lenguaje BASIC.* Sin embargo, cuando haya diferencias, añadiremos en el capítulo «Prácticas con el microordenador»: *Ojo, que esto mismo en su microordenador se dice de esta forma.* De este modo, podrá dialogar tranquilamente con su microordenador sin desagradables sorpresas y traducir al lenguaje que entiende su microordenador un programa que en BASIC estándar pueda estar escrito de forma un poco diferente.

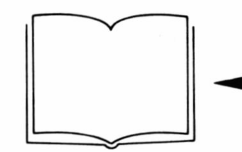
Concretamente, en el estudio de las dos partes debe proceder del modo siguiente:

— Comience siempre cada capítulo por el texto principal y continúe hasta que encuentre esta viñeta en el margen izquierdo:



Verá que en la pantalla de la viñeta aparecen unos números. Concretamente 1.1. Esto indica que debe dejar en este momento el capítulo que está estudiando, y pasar al apartado 1.1. de «Prácticas con el microordenador».

— Continúe ahora el estudio de «Prácticas con el microordenador» hasta que encuentre esta otra viñeta:



En ella se le remite de nuevo al capítulo que dejó anteriormente, para continuar donde lo interrumpió. También aquí se le indica el sitio exacto donde debe continuar. En todo caso, le recomendamos que deje siempre una señal en la página donde interrumpe el estudio. Así tendrá siempre la página localizada.

Observará el lector de esta «Enciclopedia del BASIC» que la numeración de capítulos es correlativa, iniciándose en el Tomo I de la misma y finalizando en el Tomo V y último.

Hemos creído conveniente tal continuidad, en primer lugar por el carácter secuencial de la Enciclopedia, al ser los temas que la constituyen correlativos entre sí y formando un todo inseparable; y en segundo lugar porque estimamos que ello facilita la *rápida localización de cualquier tema*, lo que, al tratarse de una obra de consulta frecuente, supone una indudable ventaja para el lector.

Parte I
BASIC

Capítulo 13
• La programación estructurada

ESQUEMA DE CONTENIDO

13.0 OBJETIVOS

Con este capítulo se finaliza un bloque de capítulos que se inicia en el capítulo 9 y que contiene una serie de instrucciones que, aunque no son estrictamente necesarias para la programación, son realmente su corazón.

En resumen las instrucciones que se han estudiado son:

La pareja FOR/NEXT que permite la realización repetitiva de las instrucciones.

Las tablas de datos son el complemento de la repetición de las instrucciones pero para los datos.

La instrucción READ, DATA y RESTORE que permite una primera aproximación a los ficheros.

Y finalmente las subrutinas que permiten dividir los programas en trozos que realizan unas funciones específicas.

En el capítulo que iniciamos se pretende dar una visión general de la programación, como el diseño de la resolución de problemas complejos que es necesario descomponer en problemas más simples para asegurarnos de que todo el mecanismo funciona.

Tratamos de dar una visión de todas las instrucciones en conjunto para saber cómo hay que encadenarlas entre sí a fin de que la resolución de los problemas sea fiable; es decir, para que se asegure al máximo la corrección de los programas.

A nuestro juicio, las instrucciones que se han estudiado se pueden combinar de la manera que uno quiera. Sin embargo, es necesario establecer una disciplina que emana de uno mismo, para poder garantizar un desarrollo rápido y seguro de los programas.

Esta disciplina surge históricamente de la necesidad de hacer programas que sean correctos y que resuelvan problemas complejos en cuya elaboración interviene un grupo de personas.

La necesidad de coordinar este grupo de personas, para que el trabajo que realiza cada una de ellas encaje con el que ha realizado otra, impone una disciplina.

Seguramente este no es su caso. Pero creemos que es imprescindible que conozca la manera de organizarse, porque le permite coordinarse a sí mismo.

De alguna manera, Ud. puede escribir un texto como más le guste. Pero hay ciertas normas que debe respetar, si el texto lo deben leer otras personas.

Las normas de circulación se establecen para poder controlar un movimiento excesivamente complejo de automóviles. Evidentemente para ir en coche no es necesario respetarlas, pero entonces la circulación es un caos. Esto también se puede aplicar a la programación. Lo que pretenden las normas que le damos es evitar que caiga en un caos lógico.

Este capítulo aborda la llamada *programación estructurada*, que podemos definir, como una *disciplina para que la programación y sus resultados, los programas, sean más fáciles de entender y de seguir*.

Tenga en cuenta que las normas que discutimos en este capítulo son mucho más amplias que el propio lenguaje de programación BASIC. Se puede decir que lo que estudiaremos nos da unos criterios para acercarnos a la resolución de problemas que, en definitiva, es el fin último de la programación.



13.1 LA PROGRAMACION ESTRUCTURADA

La idea de la programación estructurada surge a finales de la década de los sesenta como un movimiento para conseguir realizar proyectos de programación complejos.

La inicia Dijkstra, que es un ingeniero que participa en el desarrollo de los llamados sistemas operativos. Los sistemas operativos son programas muy complejos que sirven para facilitar el acceso a los ordenadores. Este concepto es especialmente aplicable a ordenadores grandes.

En un ordenador como el suyo, el BASIC constituye a la vez el lenguaje de programación y el sistema operativo; no puede hacer ninguna operación si no es mediante instrucciones BASIC.

En tipos de ordenadores algo mayores, aquellos que llevan diskettes por ejemplo, ya incorporan una serie de programas que corresponden al concepto de sistema operativo, como por ejemplo, copiar un fichero desde un diskette a otro.

Junto con Hoare y Dahl, otros analistas implicados en el desarrollo de sistemas operativos y lenguajes de programación, Dijkstra publica en 1968 un libro que se titula La Programación Estructurada.

La polémica aparece porque dicen que un lenguaje de programación no debe contener la sentencia GO TO. No es fácil imaginar en el primer momento cómo realizar programas sin la utilización del GO TO.

Ciertamente esto no es aplicable al BASIC, si se toma al pie de la letra, pues el GO TO es una instrucción específica del BASIC, pero hay que reconocer en esta polémica la intención de que no se abuse de esta instrucción.

Evitar el abuso del GO TO

Cuando se realiza un programa y se utiliza la instrucción GO TO sin ningún criterio se cae fácilmente en un ir y venir de arriba abajo que dificulta mucho el seguimiento de las operaciones que realiza el programa.

En la figura 1 se muestra un ejemplo de un programa en el que se exagera la utilización de la instrucción GO TO. El programa pretende escribir 3 veces la palabra HOLA.

Pruebe que el programa funciona correctamente. Evidentemente hay una utilización excesiva de la instrucción GO TO. El programa es muy difícil de seguir, de leer y, por lo tanto, de entender.

En la misma figura se realiza un seguimiento del programa para que se vea la dificultad de seguirlo al ir continuamente arriba y abajo.

Vea que el programa se puede escribir de esta otra manera y se harán más patentes las diferencias.

```
10 LET I=1
20 IF I>3 THEN END
30    PRINT "HOLA"
40    LET I = I + 1
50    GO TO 20
```

Se realiza el mismo trabajo con una sola instrucción GO TO y además el programa es mucho más fácil de seguir y de entender.

Este ejemplo es una exageración. Pero sin llegar a ella, se consiguen efectos similares cuando el programa es largo y aunque no se utilicen mu-

Figura 1 Ejemplo de programa con uso abusivo de la instrucción GO TO

```
PROGRAMA:               10 GO TO 40
                        20 PRINT "HOLA"
                        30 GO TO 70
                        40 LET I = 1
                        50 IF I > 3 THEN END
                        60 GO TO 20
                        70 LET I = I + 1
                        80 GO TO 50

FUNCIONAMIENTO:

          10        Se inicia el programa.
          40        Se coloca I a 1.

          50        Se pregunta si I es mayor que 3.
          60
          20        Se imprime HOLA por primera vez.
          30
          70        Se incrementa en uno a I.
          80

          50        Se pregunta si I es mayor que 3.
          60
          20        Se imprime HOLA por segunda vez.
          30
          70        Se incrementa en uno a I.
          80

          50        Se pregunta si I és mayor que 3.
          60
          20        Se imprime HOLA por tercera vez.
          30
          70        Se incrementa en uno a I.
          80

          50        Se finaliza el programa porque
                    I es mayor que tres.
```

chos GO TO, es fácil caer en este ir y venir que hace muy difícil seguir el programa.

Imagine cuando lee un libro que éste le remita al último capítulo, después al primero y así sucesivamente. Es un libro incómodo de leer. A un programa le sucede igual.

La diferencia está en que, sin darnos cuenta, si no nos imponemos una disciplina, en un programa caemos en este defecto con frecuencia.

13.1.1 ¿Qué es la estructura de un programa?

El nombre de programación estructurada sugiere algo que tiene una estructura. Estudiemos qué significa estructura en un programa.

Definición de estructura

La palabra estructura se asocia a un todo que puede dividirse en unas partes claras, que se interconectan unas con otras, para realizar funciones complejas gracias a la coordinación de todas las partes.

El ejemplo más claro y más cercano es el cuerpo humano. Es fácil distinguir partes bien diferenciadas: el esqueleto y los tejidos. Los tejidos se han especializado y dan lugar a los órganos que realizan una misión que se coordina con el resto de órganos. ¿Tiene algún sentido considerar el hígado sólo, o un cerebro sin el resto de órganos? Sólo se puede hablar de ser humano, cuando se posee la estructura de órganos completa.

Si analizamos más profundamente los órganos, establecemos que se componen de muchas células, pero que cada una tiene a su vez una estructura: Membrana celular, protoplasma, membrana nuclear y núcleo.

En definitiva, la naturaleza es un ejemplo de estructuras en la que se combinan muchos elementos para realizar misiones especializadas y que consiguen realizar la misión asignada, gracias a una coordinación de los elementos.

El hombre ha imitado esta manera de trabajar de la Naturaleza desde los tiempos más remotos, y en particular desde la Revolución Industrial, en donde el efecto de la coordinación se ha puesto de manifiesto en la construcción de máquinas complicadas.

La idea de estructurar un programa responde a la misma filosofía:

Hay que construir un todo complejo a partir de instrucciones elementales. Es posible pensar que una coordinación de todas ellas permite obtener un resultado complejo, pero ciertamente el proyecto es difícil. Es mucho mejor tomar bloques más pequeños que juntamos adecuadamente para obtener las funciones complejas que deseamos.

El bloque como una idea fundamental

La idea fundamental es, por lo tanto, la realización de unos bloques que permitan la construcción del edificio del programa con una correcta interacción entre ellos.

La palabra bloque es la palabra clave en la programación estructurada. Con el concepto de bloque se establece la unidad de construcción que permite edificar todo un programa complejo.

13.1.2 El concepto de bloque

Se entiende como *bloque* a un conjunto de instrucciones que se ejecutan desde un punto inicial, que es la *entrada al bloque*, hasta un punto final, que es la *salida del bloque*, asegurándose siempre, que si se ha iniciado el bloque, se alcanza el punto final y no se puede salir del bloque sin alcanzar este punto final.

En primer lugar un programa constituye un bloque en sí mismo. El programa se inicia, en el caso del BASIC, con el RUN y termina siempre devolviendo el control al BASIC, para realizar modificaciones o instrucciones de ejecución inmediata.

Un programa siempre es un bloque pero suele ser demasiado grande para que se tenga en cuenta y sea útil.

En el otro extremo una instrucción es también un bloque. Sólo se puede entrar en ella ejecutándola y después se inicia la instrucción siguiente. Esto es verdad para todas las instrucciones excepto precisamente el GO TO

Bloques extremos: el programa y la instrucción

La instrucción como bloque es demasiado pequeña para que tenga una utilidad para dar estructura a un programa. De todas maneras no es posible considerar ningún bloque que no contenga al menos una instrucción.

Entre estos dos bloques extremos, el programa y la instrucción de BASIC, hay muchos niveles que son precisamente los que hacen útil el concepto de bloque. Tenga en cuenta también que, a veces el considerar el programa como un bloque o el considerar una instrucción como un bloque es necesario, e incluso es lo más correcto.

En general, un bloque contiene más de una instrucción pero no contiene demasiadas, de modo que la lectura del bloque permite una rápida compresión de lo que hace. El número de instrucción es relativo y puede modificarse de acuerdo con la experiencia que tiene el programador; aunque tampoco demasiado, pues el programa lo puede utilizar un programador que no tenga ninguna experiencia.

Desde el punto de vista del BASIC, la definición del bloque puede contener cualquier instrucción, incluso la instrucción GO TO, mientras se mantenga dentro de los límites del bloque. No pueden utilizarse instrucciones GO TO que salgan fuera del bloque sin pasar por el punto final.

Considere el ejemplo de entrar una contestación a la pregunta de si se desea continuar o no un programa.

```
1000 REM Confirmación para seguir
1010 INPUT "Desea continuar (S/N)",A$
1020 IF A$ <> "N" AND A$ <> "S" THEN GO TO 1010
1030 REM Finalización del bloque
```

La descripción de bloque es: el inicio se realiza en la instrucción 1000, el final está en la instrucción 1030. La función del bloque es obtener una variable A$ con el valor de una S o una N en mayúsculas a la salida del bloque.

El bloque contiene la línea 1010 que pide la entrada de la variable A$ desde el teclado.

La línea 1020 pregunta si la variable es una N o S mayúscula. Si la respuesta es afirmativa se envía el programa a pedir de nuevo la confirmación. Cuando se alcanza la línea 1030 la variable A$ sólo puede contener una N o una S.

En muchos programas la línea 1020 es:

```
1040 IF A$ = "S" THEN GO TO 10
```

que envía al inicio del programa.

Observe que desde la línea 1000 hasta la línea 1030 constituye un bloque, pues aunque se utiliza la instrucción GO TO nunca se traspasa la finalización del bloque.

Esta confirmación se suele escribir muchas veces de la forma siguiente:

```
1000 REM Confirmación para seguir
1010 INPUT "Desea continuar (S/N)",A$
1020 IF A$ = "S" THEN GO TO 10
1030 IF A$ <> "N" THEN GO TO 1010
```

Observe que en este caso el conjunto de instrucciones no constituye un bloque. En la instrucción 1020 se envía el programa a la línea 100 si la respuesta es S y si la respuesta es N se sale por la línea 1030. El bloque tiene dos salidas.

Evidentemente el efecto es el mismo (si detrás colocamos la instrucción 1040 en el primer caso), pero de hecho no nos atenemos en el segundo caso a la disciplina de bloques. Como cualquier disciplina se puede abandonar. Pero si se relajan las condiciones, se cae fácilmente en un programa sin estructura.

La subrutina puede pertenecer a un bloque, debido al retorno automático

Un comentario aparte merece la instrucción de llamada a una subrutina que aunque sale al exterior del bloque siempre retorna a él de una manera automática.

Se puede pensar que se puede salir del bloque con una instrucción GO TO y volver al interior del bloque mediante la misma instrucción colocada en otra parte del programa.

Esta práctica no es conveniente, ya que el retorno al interior del bloque no es automático; es decir, requiere que expresamente el programador coloque la instrucción GO TO en alguna parte del programa y, por lo tanto, este GO TO está sometido a una posible revisión inadvertida por el programador cuando se consideran las funciones en la otra parte del programa.

La subrutina es el elemento más idóneo para dotar de estructura a un programa. Constituye un conjunto de instrucciones que no debemos abandonar si no es mediante una instrucción GO SUB. Sin embargo, el concepto de bloque puede usarse en cualquier trozo de programa, aunque se utiliza más frecuentemente en las subrutinas.

La ventaja que se obtiene mediante la utilización de bloque es que un programa es un conjunto de bloques que se juntan. Evidentemente, como sólo poseen una entrada y una salida, sólo se pueden juntar de una manera: El inicio de un bloque debe colocarse siempre al final de otro bloque o al inicio del programa.

13.1.3 El concepto de refinamiento

En el programa de confirmación del apartado anterior queda una duda que hay que resolver. En efecto la instrucción

```
1040 IF A$ = "S" THEN GO TO 10
```

es una sola instrucción pero que tiene dos salidas, por lo tanto, ella misma no constituye un bloque.

Observe que no es el IF, como elemento de bifurcación, el que provoca que esta instrucción no sea un bloque, sino el GO TO como instrucción de salto. En la instrucción.

```
2000 IF A<0 THEN LET A = ABS(A)
```

se utiliza el IF pero no se viola el concepto de bloque. Es decir, todas las instrucciones son bloques excepto precisamente la instrucción GO TO, como ya se ha dicho anteriormente.

Sin embargo, el efecto que deseamos del programa es que vuelva el control a la línea 100 para que se vuelva a iniciar. La única solución para no violar la regla que se ha dado es que esta instrucción pertenezca al bloque inicial del programa.

Este bloque inicial puede ser muy largo y destruirnos la visión de que un bloque es esencialmente corto.

Para que las cosas se arreglen, se utiliza el concepto de *refinamiento*. Este concepto lo que pretende, en definitiva, es modificar la definición de bloque de tal manera que en lugar de constituirse como un conjunto de instrucciones, se realiza como un conjunto de bloques.

Definición de bloque

Es decir, *un bloque es un conjunto de subbloques que se ejecutan desde un punto inicial, que es la entrada al bloque, hasta un punto final, que es la salida del bloque, asegurándose siempre de que, si se ha iniciado el bloque, se alcanza el punto final y no se puede salir del bloque sin alcanzar este punto final.*

Cada uno de los subbloques tiene un nombre que cumple una función. La especificación de este subbloque en términos de otros bloques se denomina *refinamiento* del bloque.

Naturalmente el bloque de más alto nivel es el programa entero, y el de más bajo nivel es la instrucción de BASIC en nuestro caso.

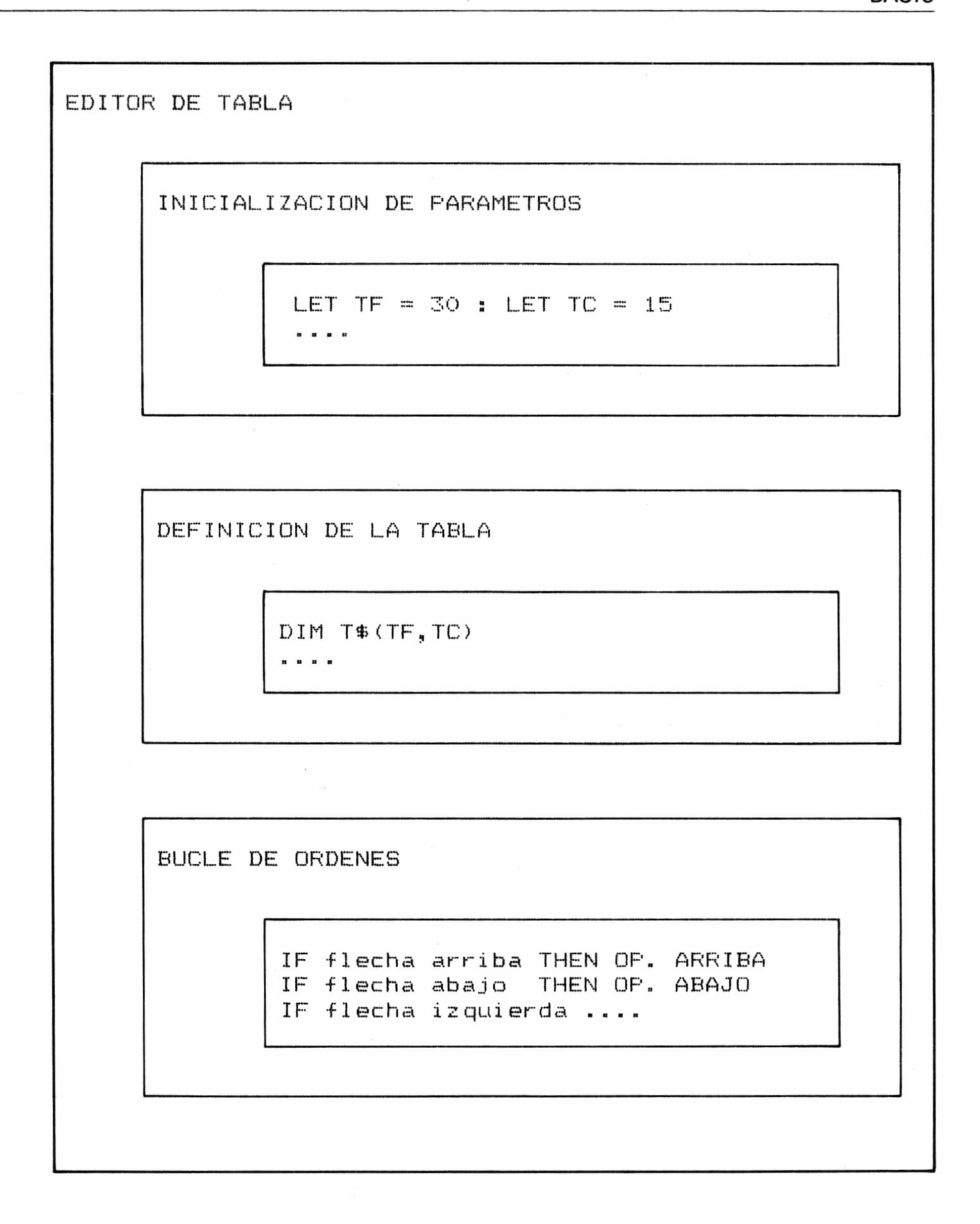
La figura 2 muestra de una manera gráfica el concepto de refinamiento y su utilización.

El cuadrado más externo representa el programa. En este caso un editor de tabla. Este programa se compone de tres subbloques, la inicialización de parámetros, la definición de la tabla y el bucle de órdenes.

Cada bloque contiene otros subbloques. En la inicialización de parámetros estos subbloques son instrucciones directamente. También ocurre lo mismo en el bloque de definición de la tabla. Este es el máximo nivel de detalle que podemos conseguir ya que sólo llegamos a las instrucciones BASIC. Ellas, a su vez, desencadenan de una manera ordenada instrucciones de máquina, pero esto para nosotros ya no es importante. Pero ciertamente hay que destacar que cada una de estas instrucciones se estructura como un bloque. Cada vez que entramos en una de ellas salimos a la siguiente instrucción (excepto el GO TO).

Observe que en el bloque del bucle de órdenes se conjugan instrucciones de BASIC y otros subbloques, como son la operación de ir arriba, la de ir abajo, etc.

Figura 2 Descomposición en bloques y con refinamiento



La programación estructurada es una disciplina que utiliza los bloques y el refinamiento

Este método permite diseñar el programa con seguridad y exactitud.

La acotación de los errores es mucho más sencilla y se puede establecer una función específica y especializada para cada bloque.

De hecho, por lo tanto, la programación estructurada es una disciplina que preconiza la utilización del bloque como elemento estructural del programa. Cada bloque a su vez se puede refinar para describir su contenido en función de otros bloques más pequeños. Este proceso de refinamiento se termina cuando alcanzamos a expresar todo el bloque en instrucciones de nuestro lenguaje (en nuestro caso el BASIC).

El bloque más natural es la subrutina. Por una parte la instrucción RETURN marca con la claridad el final del bloque y por otra parte se adapta muy bien al concepto de refinamiento ya que realiza, en general, una misión bien especificada.

En el resto del capítulo vamos a estudiar algunas estructuras típicas que permiten una organización en bloques, minimizando los efectos de utilización de la instrucción GO TO.

13.2 ESTRUCTURA SECUENCIAL

Los bloques están constituidos por subbloques. Vamos a estudiar las características de unión entre los subbloques dentro de un bloque.

Tratamos el problema de construir un bloque a partir de otros bloques, los subbloques. El problema es equivalente a dividir un bloque en subbloques.

La manera más sencilla de unir bloques es ejecutar uno detrás de otro. Cuando un bloque se compone de subbloques, que se ejecutan uno detrás de otro, se dice que posee *estructura secuencial.*

El ejemplo más sencillo de un bloque, que se estructura secuencialmente, es un cálculo. Por ejemplo, el área de la pieza irregular que se presenta en la figura 3. La figura se compone de un triángulo, de un rectángulo y de dos trapecios. Las medidas son las variables *A, B, C* y *D* que se suponen conocidas de bloques anteriores.

Las instrucciones son las siguientes:

```
2000 REM INICIO DEL BLOQUE DE CALCULO DEL AREA
2010 LET S = A*B
2020 LET S = S + A*B/2
2030 LET S = S + 2 * ( (B+D)/2 * C)
2040 REM FIN DEL BLOQUE
```

Observe que en la instrucción 2010 se calcula el área del rectángulo, como el producto de la base *(A)* por la altura *(B)* y se coloca en la variable *S.*

En la línea 2020 se calcula el área del triángulo como suma de la base *(A)* por la altura *(B)* dividido por dos; se añade este resultado al área calculada en la instrucción anterior.

Finalmente en la línea 2030 se calcula el área del trapecio que es la suma de las dos bases (*B* y *D*) divididas por dos y multiplicada por la altura *(C).* El resultado se multiplica por dos porque hay dos trapecios iguales y, finalmente, se añade al área calculada anteriormente.

Como puede deducir que la misión de este bloque es calcular el área de la pieza de la figura 3, a partir de las variables *A, B, C* y *D* para obtener el valor del área en la variable *S.* Si quiere probar el programa sólo tiene que añadirle la instrucción INPUT para entrar en los valores de las variables *A, B, C* y *D.*

La secuencia de instrucciones constituyen un bloque

Observe que el bloque es una pura secuencia de instrucciones que se ejecutan una detrás de la otra. En sí misma cada instrucción es un bloque, pues las instrucciones del BASIC son bloques.

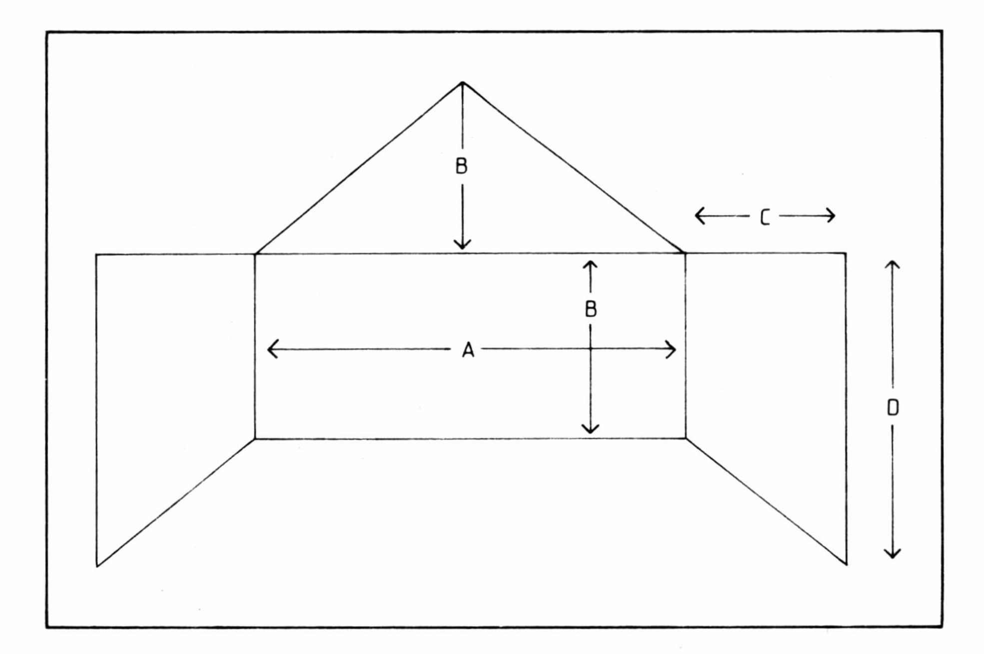


Figura 3 Pieza formada por un rectángulo, un triángulo y dos trapecios

En el apartado anterior hemos visto cómo el bloque del programa del editor de tabla se inicia con la composición secuencial de tres bloques:

— Inicialización de parámetros

— Definición de la tabla

— Bucle de órdenes

En este caso no se trata de instrucciones sencillas pero la estructura también es secuencial.

13.3 ESTRUCTURA CONDICIONAL

En el ejemplo anterior de las estructuras secuenciales hemos visto cómo un bloque se construye de bloques más pequeños que se ejecutan todos uno detrás de otro.

El siguiente esquema a considerar es cuando uno de los bloques sólo debe ejecutarse si se da una condición. Por esta razón se denomina *estructura condicional*.

Si se consideran las instrucciones del BASIC, una estructura condicional es la instrucción IF. Si no se cumple la condición que contiene detrás del IF no se ejecuta la sentencia que está detrás del THEN.

Si en lugar de hablar a nivel de instrucciones lo hacemos a nivel de bloques también la descomposición de un bloque en subbloques sucesivos puede dar origen a que un bloque se ejecute sólo si se dan ciertas condiciones.

Dos estructuras condicionales y alternativa simple y múltiple

En la programación estructurada sólo se aceptan dos estructuras como válidas; o mejor dicho, una única que en ciertos casos se reduce a una más sencilla.

Empezaremos estudiando la más sencilla para luego hablar de la más complicada, o mejor, más generalizada.

13.3.1 Alternativa simple

El caso más simple es la utilización de la frase:

si...... entonces........ en caso contrario......

esquema ya conocido en la instrucción del BASIC

IF ..... THEN ..... ELSE .....

la diferencia con la instrucción del BASIC es que en la programación estructurada cada uno de los elementos que van detrás de «entonces» (o THEN) y del «en caso contrario» (o ELSE) son un bloque, mientras que en BASIC es sólo una instrucción; es decir, un bloque del más bajo nivel de detalle.

En cambio tiene como aspecto común que la condición es una misma instrucción para el caso de la instrucción IF y de la composición del bloque.

En la figura 4 se muestra el esquema que se utiliza para simbolizar esta estructura alternativa.

En la misma figura 4 se da el esquema que se debe utilizar en BASIC para simularla. En el capítulo 7 del segundo tomo ya se habló del tema.

Puede haber bloque vacío

Naturalmente el bloque ENTONCES o el bloque EN CASO CONTRARIO puede ser vacío, es decir, que no comprenda ninguna instrucción.

Observe que a pesar de que la instrucción se descomponga en dos bloques no se viola la regla de que el bloque sólo tenga un inicio y un final.

En la figura 4 los dos bloques subordinados, es decir, el bloque ENTONCES y el bloque EN CASO CONTRARIO terminan y saltan a la instrucción de final de bloque. En la simulación en BASIC esta misión la realiza la línea 1999.

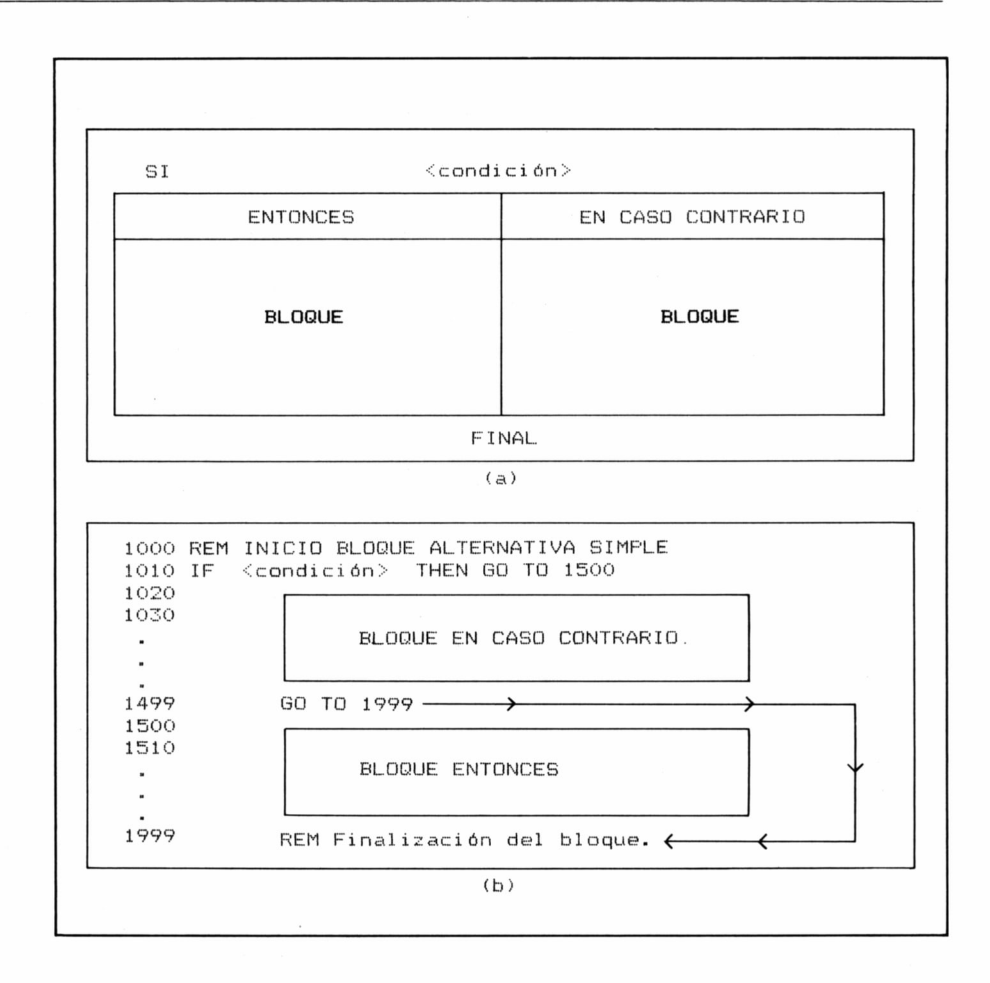
Es muy frecuente encontrar estas construcciones en el interior de las subrutinas. En este caso se utiliza el RETURN como final del bloque y se puede colocar a veces en posiciones que no son claramente final de bloque.

Por ejemplo, en la subrutina de ir hacia la izquierda con el cursor en el editor de tabla. La construcción es la siguiente.

```
8000 REM IZQUIERDA
8010 IF C= 1 THEN RETURN
8020 LET C = C -1 : LET CP = CP -1
8030 IF CP > 0 THEN RETURN
8040 LET Q= F-FP+1 : LET P=C : GO SUB 9500
8050 LET CP = CP + 1
8060 RETURN
```

En las líneas 8010 y 8030 hay un RETURN después de una estructura condicional que actúa como bloque ENTONCES.

Figura 4 a) estructura de alternativa simple con bloques; b) Simulación en BASIC



De hecho al ser subrutina corta se entiende perfectamente que este RETURN actúa como final de bloque.

Puede escribirse este bloque de la manera siguiente:

```
8000 REM IZQUIERDA
8010 IF C= 1 THEN GO TO 8060
8020 LET C = C -1 : LET CP = CP -1
8030 IF CP > 0 THEN GO TO 8060
8040 LET Q= F-FP+1 : LET P=C : GO SUB 9500
8050 LET CP = CP + 1
8060 RETURN
```

Se han sustituido los RETURN de las líneas 8010 y 8030 por un GO TO a la línea 8060 que es el RETURN que marca el final de la subrutina. No hace falta que escriba ahora estos bloques, salvo que los utilice dentro de un programa.

La utilización de una forma u otra es una cuestión de gusto. En ningún caso la subrutina queda no estructurada.

13.3.2 Alternativa múltiple

La generalización del caso anterior consiste en que en lugar de utilizar dos vías alternativas se utilice más de una.

En el caso de alternativa simple se ejecuta el bloque del ENTONCES o bien el bloque de EN CASO CONTRARIO, pero nunca los dos o ninguno.

La alternativa múltiple y una generalización de la alternativa simple

Ahora en el caso de la alternativa múltiple se aumenta el número de vías posibles a tomar. En lugar de haber dos bloques hay más de dos. Pero cuando se ejecuta esta estructura condicional sólo se pasa por uno de los subbloques que corresponden a las distintas alternativas.

La figura 5 muestra cómo se construye el bloque. En primer lugar se evalúa una expresión que da un resultado numérico. Según el valor de este resultado se ejecuta uno de los bloques.

Figura 5 Esquema del bloque de alternativa múltiple

SEGUN <expresión>

CASO 2	CASO 127	CASO 26	...	OTRO VALOR
BLOQUE 1	BLOQUE 2	BLOQUE 3	...	BLOQUE n

FINAL

En la figura 5 se muestra que si el resultado de evaluar la expresión es 2 se pasa por el subbloque 1, en el caso de que sea 127 se pasa por el subbloque 2, en el caso de que sea 26 se pasa por el subbloque 3 y así sucesivamente.

Es importante darse cuenta que el último bloque consiste en cualquier otro valor que no esté en los casos anteriores.

Estos subbloques pueden a su vez ser vacíos, es decir, no contienen ninguna instrucción.

Cuando se finaliza el bloque sólo se ha pasado por uno de los subbloques correspondientes a las distintas alternativas.

Para programar en BASIC es necesario adaptarse a esta disciplina del modo que se indica en la figura 6.

La línea 1010 calcula una expresión cuyo valor se asigna a la variable *O*. (Se ha numerado a partir de 100 para indicar que se está en el interior de un programa más largo).

La línea 1020 se pregunta si la variable *O* es distinta al valor 2. En caso afirmativo se salta a la línea 1300. Observe que se pregunta si la opción es distinta de 2. Si la respuesta es negativa es que *O* vale 2, entonces el programa prosigue por las líneas 1030 y sucesivas que corresponden al subbloque 1 tal como indica la figura.

Cuando finaliza el bloque 1 se encuentra la instrucción 1299, la numeración se ha elegido para indicar que pueden haber muchas instrucciones, que es un GO TO a 1999 que es el final del bloque.

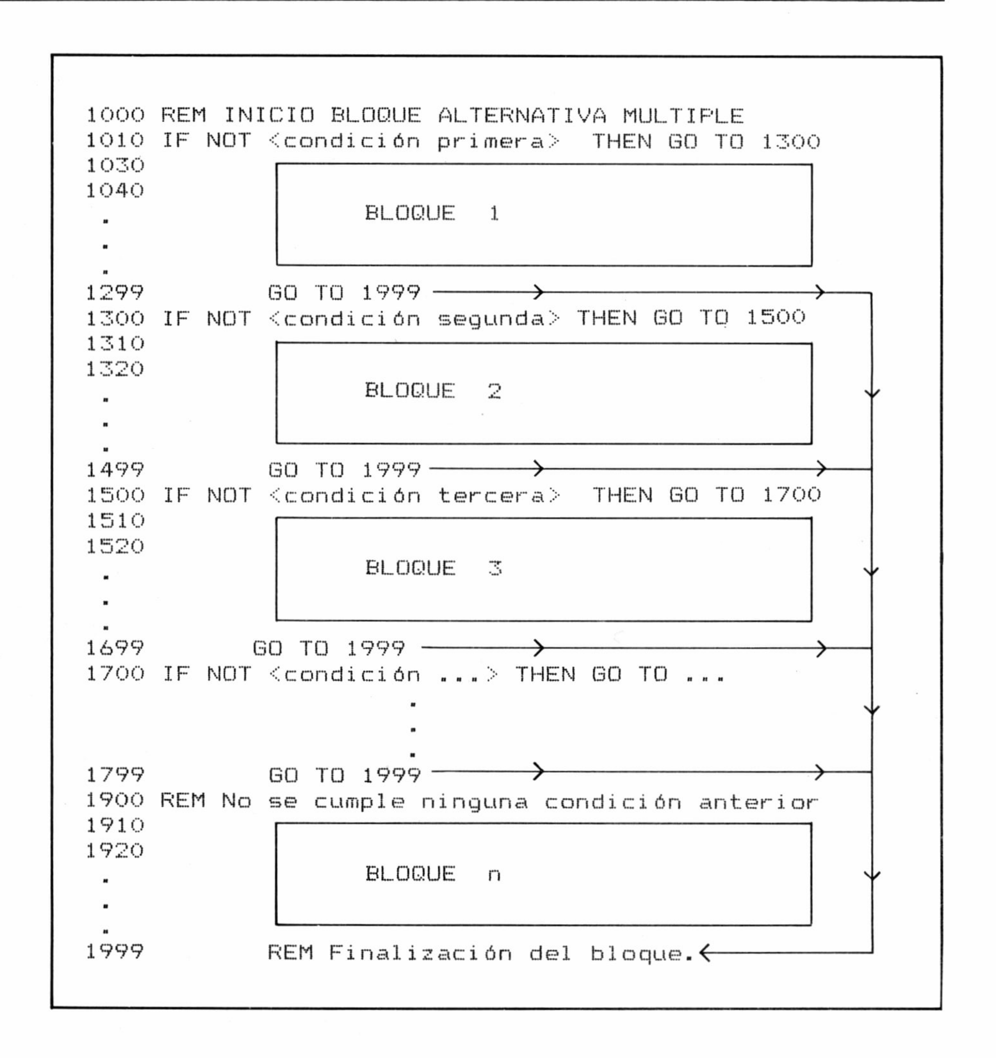


Figura 6 Simulación en BASIC de un bloque de estructura condicional de alternativa múltiple

Observe que si la opción *O* no es ninguna de las alternativas señaladas en cada uno de los IF correspondientes se alcanza la línea 1900 en donde se inicia el subbloque correspondiente a OTRO VALOR, tal como se indica.

En conclusión, para simular un bloque de estructura condicional de alternativa múltiple es necesario evaluar una variable, que puede ser numérica o textual (en la figura es numérica). A continuación se coloca la pregunta de si la variable es distinta a la opción.

La alternativa múltiple se basa en la evaluación de una variable

En caso afirmativo, se salta a una nueva pregunta. En caso contrario, se realiza el bloque correspondiente a la alternativa, y una vez se ha finalizado el subbloque se encuentra un GO TO que lo manda al final del bloque condicional.

En general, para evitar que los problemas de este tipo sean muy largos, los subbloques consisten en una llamada a subrutina y, por lo tanto, en el lugar donde se produce este bloque condicional, el subbloque consiste en una línea.

Este tipo de estructuras se aplica frecuentemente a los bucles de órdenes para realizar diversas operaciones en el programa.

El diseño para un programa que realiza diversas acciones según la tecla que pulsamos es un ejemplo de un bucle condicional con alternativa múltiple.

En un editor de tabla, las órdenes son las siguientes:

S(ubir)	Desplazar el cursor hacia arriba.
B(ajar)	Desplazar el cursor hacia abajo.
D(erecha)	Desplazar el cursor hacia la derecha.
I(zquierda)	Desplazar el cursor hacia la izquierda.
P(rimero)	Borde superior izquierdo de la tabla.
U(ltimo)	Borde superior derecho de la tabla.
F(inalizar)	Finalizar.

Se toman las iniciales de las palabras de la izquierda para desencadenar la operación descrita por las frases de la derecha.

Se trata de un bloque condicional de alternativa múltiple porque hay que emprender una acción para cada término.

Por otra parte, si no se pulsa ninguna de las letras que son las iniciales de las órdenes el programa debe dar un mensaje de error. Este constituye el bloque de OTRO VALOR que se ha mencionado.

El programa es el siguiente:

```
100 REM Bucle de órdenes.
110 LET A$ = INKEY$ : IF A$ = "" THEN GO TO 110
120 IF A$<> "S" THEN GO TO 150
130     PRINT "Desplazar el cursor hacia arriba"
140     GO TO 400
150 IF A$<> "B" THEN GO TO 180
160     PRINT "Desplazar el cursor hacia abajo"
170     GO TO 400
180 IF A$<> "I" THEN GO TO 210
190     PRINT "Desplazar el cursor hacia la izquierda"
200     GO TO 400
210 IF A$<> "D" THEN GO TO 240
220     PRINT "Desplazar el cursor hacia la derecha"
230     GO TO 400
240 IF A$<> "P" THEN GO TO 270
250     PRINT "Colocarse en el primer elemento de la tabla"
260     GO TO 400
270 IF A$<> "U" THEN GO TO 300
280     PRINT "Colocarse en el último elemento de la tabla"
290     GO TO 400
300 IF A$<> "F" THEN GO TO 330
310      END
320      GO TO 400
330 PRINT "Orden errónea"
400 REM Finalización del bloque condicional.
500 GO TO 100
```

En lugar de colocar las acciones se ha colocado una instrucción de PRINT para probar el funcionamiento. Más tarde cuando el funcionamiento sea correcto se sustituyen por las instrucciones de la acción, o, lo que es más frecuente, se envía a ejecutar la subrutina correspondiente.

Se utiliza como expresión de cálculo la función INKEY$ que devuelve el carácter que se ha pulsado en el teclado. En este caso se ha elegido una variable textual en vez de numérica tal como se ha mostrado en el esquema general.

Observe que el bloque de finalización contiene un END, que termina el programa y que la instrucción 500 no pertenece al bloque. Su misión es iniciar otra vez el bloque una vez se ha salido de él.

El GO TO 400 después de la instrucción END no tiene sentido en este programa, pero se coloca para cumplir fielmente con el esquema desarrollado. Por otra parte, piense que a veces es necesario terminar el programa después del bloque condicional, entonces no es posible colocar un END en el interior. Se estudiará de nuevo este ejemplo más adelante, en las estructuras iterativas.

También debe tener en cuenta que a veces dos valores distintos pueden tener la misma acción. Por ejemplo, si además de la letra P, para ir al primer lugar de la tabla, se quiere utilizar el número 1 entonces se tiene que dos casos corresponden a la misma acción, es lo mismo pulsar la letra P que el 1.

En esta circunstancia se utilizan las preguntas del IF más complicadas. En este caso la línea 240 debe sustituirse por

```
240 IF A$<> "P" AND A$<> "1" THEN GO TO 270
```

que realiza una pregunta doble: si la variable *A$* es distinta de P y de 1 es cuando se salta el bloque correspondiente.

Recuerde que esta es la forma de negación de que *A$* es igual a P o a 1. Vea las leyes de Morgan del capítulo 6, en el segundo tomo.

13.3.3 La instrucción ON... GO TO... y ON... GO SUB...

El BASIC dispone de dos instrucciones adaptadas a la estructura condicional de alternativa múltiple.

Síntesis general

Estas instrucciones son muy similares en su manera de escribirse. La forma general es la siguiente:

ON J GO TO <línea 1>, <línea 2>...<línea N>
ON J GO SUB <línea 1>, <línea 2>... <línea N>

la variable *J* es la variable de control que puede tener un valor cualquiera.

Detrás del ON *J* aparece el GO TO o GO SUB según el caso.

Finalmente aparece una lista de líneas del programa separadas por coma.

La acción de esta instrucción es enviar el programa a la línea 1 si el valor de *J* es 1, a la línea 2 si el valor de *J* es 2, envíar a la línea 3 si el valor de *J* es tres y así sucesivamente.

En el caso de que *J* sea inferior o igual a cero, o bien mayor que N no se bifurca el programa a ninguna parte y se sigue en la línea siguiente.

La diferencia entre utilizar el GO TO o el GO SUB es que en el primer caso se transfiere el control a la línea mencionada como en el caso del GO TO normal.

En el caso del GO SUB se transfiere el control del programa a la línea mencionada como una subrutina, es decir, cuando retorna se alcanza la sentencia siguiente al ON... GO SUB.

La figura 7 da el esquema general para realizar un bloque condicional de estructura múltiple cuando se usa una instrucción ON ... GO TO.

La diferencia de este caso respecto a la construcción general del GO TO mencionada antes son dos:

a) Los valores de la variable de control para realizar la bifurcación deben ser seguidos, es decir, debe valer 1, 2, 3...

Si el valor está fuera de márgenes se sigue en la instrucción posterior al ON... (GO SUB o GO TO)

b) El bloque para un valor que cae fuera de estos valores (es decir, en el caso de OTRO VALOR) está inmediatamente debajo de la instrucción ON... GO TO.

Para ilustrar el funcionamiento de esta instrucción, el ejemplo del bloque de órdenes se escribe así:

```
 10 LET O$="SBDIPUF"
100 REM Bucle de órdenes.
110 LET A$ = INKEY$ : IF A$="" THEN GO TO 110
120 FOR I = 1 TO 7
130 IF MID$(O$,I,1) = A$ THEN GO TO 150
140 NEXT I
150 ON I GO TO  180,200,220,240,260,280,300
160      PRINT "Orden errónea"
170      GO TO 400
180      PRINT "Desplazar el cursor hacia arriba"
190      GO TO 400
200      PRINT "Desplazar el cursor hacia abajo"
210      GO TO 400
220      PRINT "Desplazar el cursor hacia la izquierda"
230      GO TO 400
240      PRINT "Desplazar el cursor hacia la derecha"
250      GO TO 400
260      PRINT "Colocarse en el primer elemento de la tabla"
270      GO TO 400
280      PRINT "Colocarse en el último elemento de la tabla"
290      GO TO 400
300      END
310      GO TO 400
400 REM Finalización del bloque condicional.
500 GO TO 100
```

Observe que en esta manera de construir el programa primero se utiliza la cadena de caracteres O$ que contiene las letras válidas como órdenes en el orden que requieren los bloques en ejecución (línea 10).

En la línea 110 se obtiene el carácter pulsado y se coloca en la variable A$.

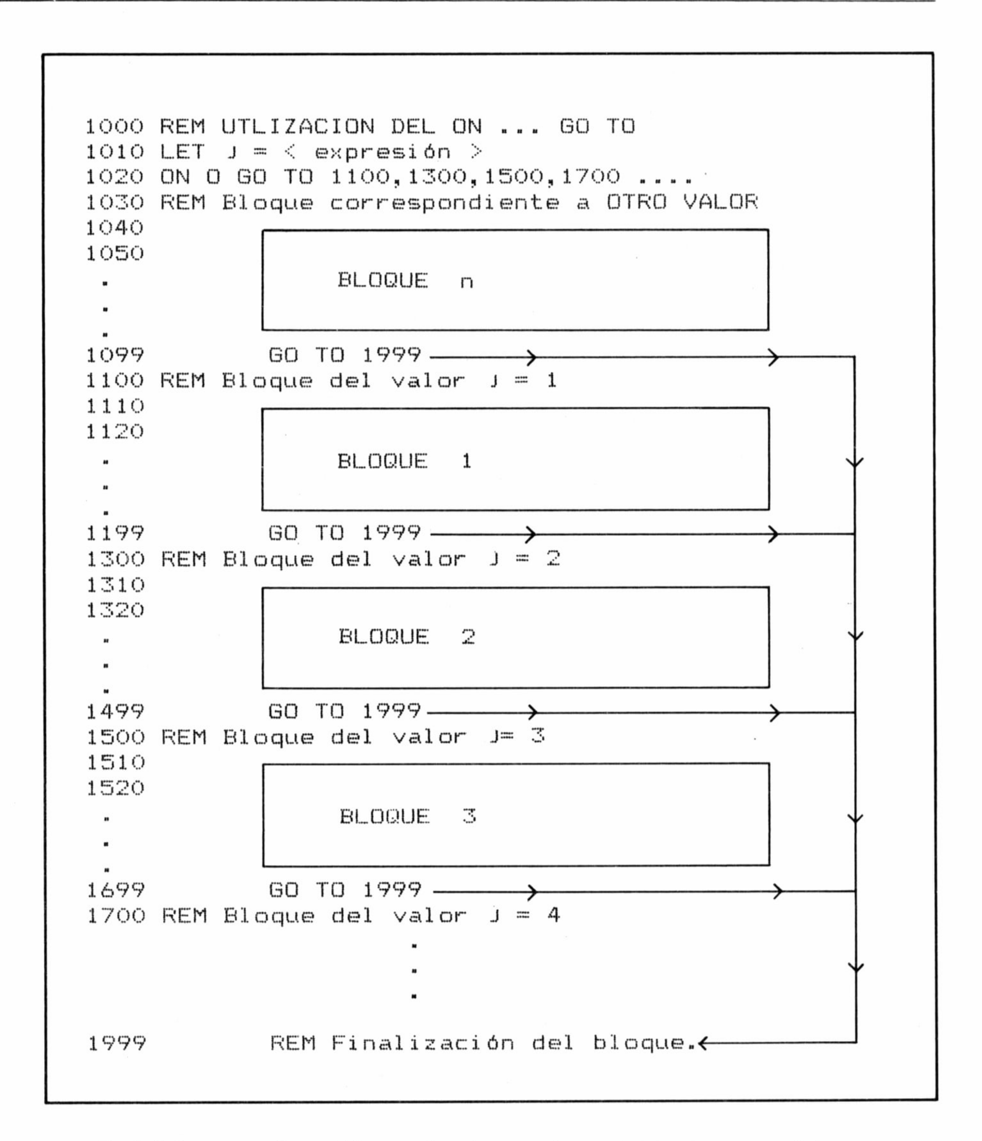


Figura 7 Utilización de la instrucción ON... GO TO... para las estructuras condicionales de alternativa múltiples

Se inicia a continuación un bucle con la variable de control *I* y se pregunta en el interior del bucle (línea 130) si corresponde al carácter i-ésimo de la cadena de caracteres. Si la respuesta es afirmativa se envía el control a la línea 150. En caso de que no sea afirmativa se alcanza el final del bucle.

Si el carácter pulsado no se corresponde con ninguna de las órdenes se alcanza el final de bucle con el valor de la variable de control igual a 8.

La línea 150 es la instrucción ON... GO TO... que tiene solamente 7 líneas detrás. Se produce el envío si *I* está comprendida entre 1 y 7; en el caso de que valga 8 se alcanza la línea 160 que imprime el mensaje de orden errónea.

Por lo demás, la estructura muestra que cada uno de los bloques finaliza con un salto a la línea 400 que es la finalización del bloque condicional.

Si, como se ha dicho, el resultado final es ir a realizar una subrutina se puede pensar en utilizar el GO SUB.

En este caso, la línea 150 se ha de escribir como:

```
150 ON I GO SUB  8000,8100,8200,8300,8400,8500,9000
```

en donde, se supone que las direcciones especificadas son las de las subrutinas que atienden a cada una de las acciones.

Esta simple sustitución es errónea. ¿Por qué?

Cuando se vuelve de la subrutina se va a la instrucción siguiente, es decir, se inicia la línea 160 que corresponde a la de orden errónea. En estos casos hay que ser mucho más severo con los valores de la variable de control ya que el programa acaba en la instrucción siguiente después de ir a la subrutina.

Una posible corrección de este efecto es:

```
 10 LET O$="SBDIP1UF"
100 REM Bucle de órdenes.
110 LET A$ = INKEY$ : IF A$="" THEN GO TO 110
120 FOR I = 1 TO 8
130 IF MID$(O$,I,1) = A$ THEN GO TO 150
140 NEXT I
150 ON I GO SUB  8000,8100,8200,8300,8400,8400,8500,9000,9900
160 REM Finalización del bloque.
200 GO TO 100
```

Se ha añadido la subrutina 9900 que trata el caso de que la orden es errónea. El valor de I sólo puede valer 8 cuando sale del bucle o de 1 a 7 si sale desde el interior del bucle. Por lo tanto, la condición de OTRO VALOR no se da nunca en esta construcción.

Como puede observar, la instrucción ON... GO SUB... da unas estructuras condicionales de altenativa múltiple más limpias y fáciles de seguir. Esto es debido sin duda, al poder de estructuración que tiene la subrutina.

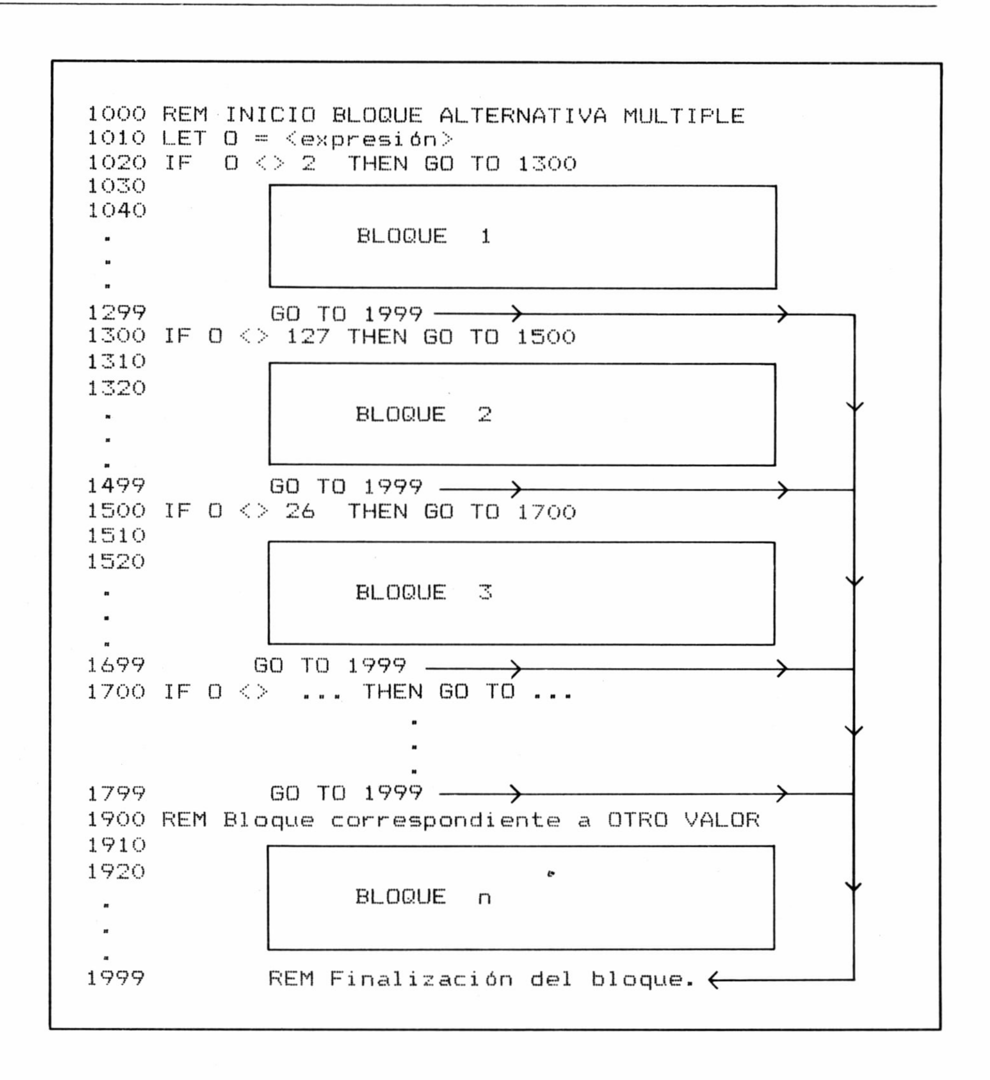
Si se quiere que dos teclas den lugar a la misma acción también puede realizarse con la instrucción ON... GO TO u ON... GO SUB, ya que puede repetirse el número de línea en la lista de direcciones.

Si consideramos el mismo ejemplo de antes en el que el 1 es una forma válida de indicar el primero de la tabla. Basta realizar los cambios que se indican a continuación:

```
 10 LET O$="SBDIPUF"
100 REM Bucle de órdenes.
110 LET A$ = INKEY$ : IF A$="" THEN GO TO 110
120 FOR I = 1 TO 7
130 IF MID$(O$,I,1) = A$ THEN GO TO 150
140 NEXT I
150 ON I GO SUB  8000,8100,8200,8300,8400,8500,9000,9900
160 REM Finalización del bloque.
200 GO TO 100
```

En la línea 10 se ha añadido un 1 detrás de la P. En la línea 120 se ha modificado el valor superior del bucle a 8 y finalmente en la línea 150 se ha repetido el valor de 8400 que corresponden a los casos de pulsar la tecla P o la tecla 1.

Figura 8 Empleo en BASIC de una estructura condicional de alternativa múltiple guiada por condición



13.3.4 Condiciones múltiples

Para finalizar una estructura condicional múltiple se puede construir no con un único valor sino con muchas condiciones que juegan el papel del valor.

La figura 8 muestra el esquema general cuando las condiciones son múltiples.

En la figura se ha eliminado la obtención de un valor y en cambio en cada instrucción IF se pregunta si *no* se da una condición. En caso afirmativo se envía a considerar la condición siguiente. En caso de respuesta negativa, es decir, que se cumple la condición, se entra en el bloque.

El último bloque, como de costumbre, consiste en aquel que se alcanza si no se cumple ninguna de las condiciones anteriores.

Para aclarar este esquema general, considere el problema de dar una nota final a un alumno de una asignatura, sabiendo que ha realizado un examen oral y un examen escrito ambos puntuados sobre 10.

Para aprobar es necesario que la media de los dos exámenes sea su-

perior a 5 y que la nota del examen escrito no sea inferior a 4. Además si la nota media es superior a 8 debe recibir mención especial.

La entrada de datos es un valor correspondiente a cada uno de los exámenes, que colocamos en las variables *O* y *E*.

El problema se descompone en el cálculo de la media, por una parte, y la decisión de si se ha de suspender, aprobar o dar mención.

La media que se coloca en la variable *M* se calcula como semisuma de las variables *O* y *E*, es decir en BASIC se expresa como:

```
LET M = (O + E) /2
```

La manera de describir estos resultados es mediante la siguiente tabla.

	0 < = M < 5	5 < = M < = 8	8 < M
0 < = E < 4	SUS.	SUS.	SUS.
4 < = E	SUS.	APR.	MENCION

Cada resultado corresponde a cada una de las preguntas siguientes:

— SUSPENSO Si E < 4 OR M < 5.

— APROBADO Si 4 <= E AND 5 <= M AND M <= 8.

— MENCION Si 4 <= E AND 8 < M

El programa debe estructurarse como un bloque condicional de estructura múltiple con los tres subbloques que asigna el resultado Suspenso, Aprobado y Mención.

Ahora el problema es que no tenemos tres valores sino que se dan tres condiciones descritas más arriba.

El programa es el siguiente:

```
100 INPUT "Nota Oral:", O
110 INPUT "Nota Escrito:",E
120 REM Cálculo de la media
130 LET M = ( O+E)/2
140 REM Inicio del bloque condicional.
150 IF NOT ( E<4 OR M<5 ) THEN GO TO 200
160     PRINT "SUSPENSO"
170     GO TO 400
200 IF NOT ( 4<= E AND 5<=M AND M<=8 ) THEN GO TO 300
210     PRINT "APROBADO"
220     GO TO 400
300 IF NOT ( 4<= E AND 8<M) THEN GO TO 400
310     PRINT "MENCION"
400 REM Finalización del bloque
500 GO TO 100
```

Las condiciones deben ser excluyentes

Realmente el programa no está muy cuidado ya que se pueden introducir valores erróneos.

Sin embargo, si los valores introducidos son correctos, es decir, entre 0 y 10 los resultados son también correctos.

El programa ilustra la construcción a partir de condiciones antes que valores.

La única norma que hay que observar es que las condiciones sean excluyentes, es decir, no se puedan dar dos simultáneamente ya que entonces el resultado del programa depende del orden en que se pregunten.

Las reglas para demostrar que las condiciones son excluyentes no son fáciles de dar. Se recomienda, por lo tanto, utilizar este tipo de estructura con mucha precaución.

RESUMEN

La posibilidad de escribir programas largos con una utilización abusiva de la instrucción GO TO obliga a considerar la estructura del programa.

La estructura de un programa es la manera de dividir el programa en partes claras, que se interconectan unas con otras, para realizar funciones complejas mediante la coordinación de todas las partes.

La estructura de un programa se consigue mediante el concepto de bloque.

Un bloque es un conjunto de instrucciones que se ejecutan desde un punto inicial, que es la entrada del bloque, hasta un punto final, que es la salida del bloque, asegurándose siempre que, si se ha iniciado el bloque, se alcanza el punto final y no se puede salir del bloque sin alcanzar este punto final.

Un programa entero es un bloque. Una instrucción del BASIC es un bloque, excepto para la instrucción GO TO. Estos son el bloque más grande y el más pequeño que disponemos en la construcción de un programa.

La programación estructurada es una disciplina que se impone uno mismo para trabajar con bloques de tamaño intermedio entre el mayor, el programa, y el menor, la instrucción.

El propio lenguaje BASIC lleva incorporado el concepto de bloque en el concepto de subrutina. En general, la subrutina, en BASIC y en otros lenguajes es el elemento idóneo para estructurar un programa.

Un programa es mucho más fácil de seguir si se estructura en bloques. Cada bloque posee una sola entrada y una sola salida; luego, necesariamente, la unión de estos bloques consigue una unidad de lectura y ejecución del programa muy elevada.

El hecho de descomponer un bloque en subbloques que cumplan las mismas condiciones define el concepto de refinamiento.

Para definir entonces un bloque es necesario cambiar el término instrucciones por subbloques.

La definición es la siguiente:

Un bloque es un conjunto de subbloques que se ejecutan desde un punto inicial, que es la entrada del bloque, hasta un punto final, que es la salida del bloque, asegurándose siempre que si se ha iniciado el bloque se alcanza el punto final y no se puede salir del bloque sin alcanzar este punto final.

Al hablar del bloque se puede hablar de dos maneras: como entidad que realiza una función o como un conjunto de subbloques que indican cómo se realiza la función.

Cada bloque se puede refinar a su vez para describir cómo realiza la función en términos de otros bloques más pequeños. El proceso de refinamiento termina cuando alcanzamos a expresar todo el bloque en instrucciones de nuestro lenguaje.

La composición de los bloques adquiere diversas formas según el tipo de operación a realizar.

La composición secuencial se da cuando un bloque se descompone o un bloque se compone de subbloques que se ejecutan uno detrás de otro. Todos los bloques se ejecutan una vez y necesariamente sólo una vez.

La composición condicional se da cuando un bloque se descompone en bloques de los cuales sólo se ejecuta uno y sólo uno.

La estructura condicional se basa en la pregunta de una condición que bifurca el flujo del programa hacia un subbloque determinado, que es el que se ejecuta en estas condiciones.

El bloque más sencillo es el de alternativa simple, es decir, cuando la condición sólo puede ser cierta o falsa. En cada uno de estos casos se ejecuta un bloque u otro.

Los bloques más complicados son los que tienen múltiples alternativas. Entonces se utiliza un valor para distribuir el control a uno de los bloques subsidiarios.

El BASIC dispone de las instrucciones

ON (variable) GO TO (línea1), (línea2), ... (línean)

y

ON (variable) GO SUB (línea1), (línea2), ... (línean)

Aquí los valores de la variable son correlativos de 1 a n.

Finalmente, en lugar de utilizar el valor de una variable se puede utilizar un conjunto de condiciones para distribuir el control a un bloque u otro.

En este caso se debe asegurar que las condiciones son excluyentes. Para ello es necesario una demostración lógica, si las condiciones son muy complicadas. Por esta razón se debe utilizar este caso con precaución.

EJERCICIOS DE AUTOCOMPROBACION

Complete las frases siguientes:

1. La utilización abusiva de la instrucción GO TO deshace la ..... del programa.

2. El .... es el elemento que estructura un programa.

3. El bloque mayor es el ......

4. El bloque menor es la ......

5. La .... es la mejor manera de conseguir bloques.

6. La descomposición de un bloque en subbloques se denomina ....

7. La misión de un bloque es realizar una .... completa.

8. La composición .... es aquella que descompone el bloque en subbloques para que todos se ejecuten una vez y sólo una vez.

9. La composición condicional es aquella que descompone el bloque en subbloques para que se ejecute .... una vez y sólo una vez.

10. La composición condicional es de dos tipos de alternativa simple y de alternativa ....

Rodee con un círculo la letra que corresponde a la alternativa correcta.

11. La programación estructurada es

    a) Un lenguaje.

    b) Utilizar sólo unas instrucciones del BASIC.

    c) Eliminar la instrucción GO TO de los programas.

    d) Una disciplina de programación.

12. La subrutina es la mejor manera de realizar bloques en BASIC, porque.

    a) Se puede situar aparte.

    b) Tiene una sola entrada y una sola salida.

    c) Eliminan la utilización del GO TO.

    d) Está en todos los lenguajes.

13. El concepto de refinamiento en subbloques permite expresar funciones.

    a) Sin conocer como se realizan.

    b) Con menos instrucciones.

    c) Con subrutinas.

    d) Sin utilizar GO TO.

14. Todas las instrucciones en BASIC son bloques excepto la instrucción

    a) GO TO.

    b) GO SUB.

    c) IF... THEN...

    d) ON... GO SUB...

15. Cuando se juntan bloques de manera secuencial para componer un bloque mayor, al ejecutar el bloque mayor cada subbloque se ejecuta

    a) Tres veces.

    b) Cero veces.

    c) Una sola vez y sólo una.

    d) O puede ejecutarse una vez.

16. ¿Qué instrucción de las siguientes no genera una estructura condicional?

    a) IF... THEN...

    b) FOR... NEXT.

    c) ON... GO TO...

    d) ON... GO SUB...

17. ¿Qué regla deben cumplir las condiciones que sirven para bifurcar en una estructura condicional de alternativa múltiple?

    a) Ser excluyentes; si se cumple una no debe cumplirse otra.

    b) Deben cumplirse todas a la vez.

    c) No debe cumplirse ninguna.

    d) Se deben cumplir algunas a la vez.

18. El bloque vacío, es decir, que no contiene ninguna instrucción no se admite en la composición

    a) Secuencial.

    b) Alternativa simple.

    c) Alternativa múltiple por valor.

    d) Alternativa múltiple por condiciones.

19. En la instrucción ON... GO TO... cuando la variable de control es cero o mayor que el número de líneas listadas en la parte del GO TO ocurre

    a) Un error.

    b) Bifurcación a la última línea de la lista.

    c) Bifurcación a la línea siguiente de la instrucción.

    d) Finalización del programa.

20. En la instrucción ON... GO SUB... cuando la variable de control es cero o mayor que el número de líneas listadas en la parte del GO SUB ocurre

    a) Un error.

    b) Bifurcación a la última línea de la lista.

    c) Bifurcación a la línea siguiente de la instrucción..

    d) Finaliza el programa.

13.4 ESTRUCTURA DE REPETICION E ITERATIVA

En las estructuras que se han estudiado cada subbloque se ejecuta al máximo una vez. En la secuencial todos se ejecutan una vez y sólo una vez, y en la condicional de todos los subbloques que la componen, sólo se ejecuta un bloque una sola vez.

Vamos a estudiar ahora los tipos de estructura que permiten repetir un bloque más de una vez.

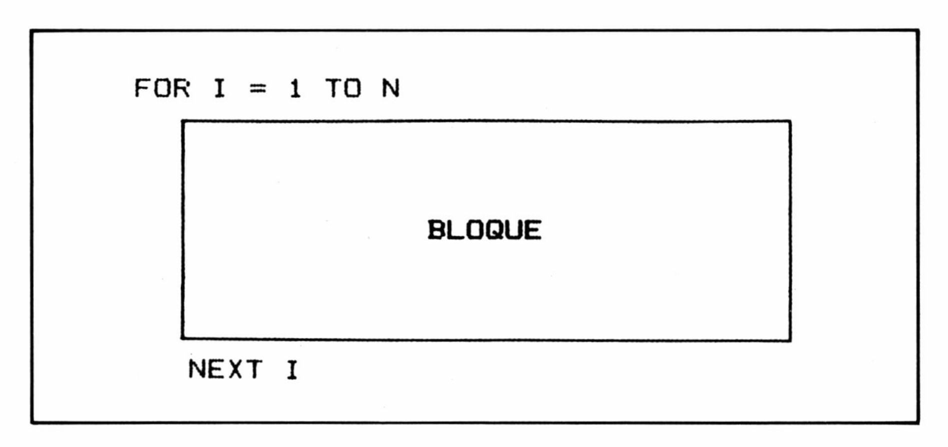


Figura 9 Composición repetitiva

Empezaremos con el más simple que es la estructura de repetición. Al entrar en esta estructura sabemos con certeza cuántas veces vamos a repetir un bloque.

La pareja FOR/NEXT es la estructura repetitiva en BASIC

La instrucción de BASIC que representa esta estructura es la instrucción FOR junto con la instrucción NEXT. Explicada de una manera general, la instrucción FOR repite tantas veces como es necesario, según las variables de control, el bloque de instrucciones que lo componen. Naturalmente, en el supuesto de que las instrucciones comprendidas entre el FOR y el NEXT cumplan con la definición de bloque, es decir, que una vez iniciado debe salirse por el final.

El esquema correspondiente está en la figura 9. Observe que por la definición de bloque la estructura es perfectamente repetitiva, es decir sabemos de antemano las veces que se ejecuta.

Se debe insistir en que si el concepto de bloque es el que se aplica a las instrucciones entre el FOR y el NEXT no es posible salir de él más que por el final y, por lo tanto, no podemos salir fuera del bucle generado por las instrucciones FOR y NEXT.

Debido a la profundidad con que se ha tratado el tema en el capítulo 9 del tomo anterior, no es necesario dar más ejemplos.

La estructura más interesante es la iterativa. La diferencia respecto a la repetitiva es que no se conoce con exactitud el número de veces que debe repetirse el subbloque.

Para ilustrar la diferencia entre repetición e iteración considere los dos programas siguientes:

1. Debe imprimir varias veces la frase «Hola que tal». Por ello se entra el número de veces que deseamos que se repita el saludo y se ejecuta a continuación en una instrucción FOR para repetir el saludo las veces que hemos entrado.

```
10 INPUT "Número de veces:", N
20 IF N <= 0 THEN END
30 FOR I = 1 TO N
40       PRINT "Hola que tal"
50       NEXT I
60 GO TO 10
```

El bloque de estructura repetitiva se inicia en la línea 30 con la instrucción FOR y finaliza en la 50 con la instrucción NEXT. El subbloque que se repite consta sólo de la instrucción PRINT de la línea 40.

2. El programa debe averiguar si la palabra que le entramos corresponde a las tres letras iniciales de un mes. Se trata de un problema de búsqueda en una tabla.

```
 10 REM Definición de la tabla de meses.
 20 DIM M$(12)
 30 FOR I = 1 TO 12 : READ M$(I) : NEXT I
 40 DATA "ENE","FEB","MAR","ABR","MAY","JUN"
 50 DATA "JUL","AGO","SEP","OCT","NOV","DEC"
 60 REM Final de definición de la tabla.
100 INPUT "Iniciales del mes:",A$
110 REM Inicio de la búsqueda en una tabla.
120 FOR I = 1 TO 12
130       IF A$ = M$(I) THEN GO TO 200
140       NEXT I
150 PRINT "Iniciales erróneas"
160 GO TO 100
200 PRINT "Se trata del mes ";I
210 GO TO 100
```

De las instrucciones 10 a la 60 se rellena la tabla *M$()* con los valores de las iniciales del mes.

La línea 100 lee en la variable *A$* una cadena de caracteres cualquiera.

Desde la línea 110 hasta la 210 se comprueba si esta cadena entrada corresponde a las iniciales del mes.

Si considera ahora el FOR de la línea 120 se puede pensar que se inicia un bloque de repetición que se ejecuta 12 veces. Observe que esta conclusión es incorrecta ya que en la línea 130 se puede salir del bucle de repetición si las letras entradas coinciden con las letras de algún mes. En este caso el control de ejecución del programa se envía a la línea 200.

Las conclusiones que se sacan de este ejemplo son dos:

1. El par de instrucciones FOR/NEXT no constituyen un bloque. Hay dos salidas: una por la instrucción 150 y otra por la instrucción 130 hacia la línea 200.

2. No se puede construir un bloque de repetición porque no se sabe cuántas veces se repite la línea 130. Recuerde que una instrucción en BASIC se puede considerar un bloque siempre que no intervenga el GO TO.

¿Cómo se construye un esquema de bloques en este caso? A continuación damos el programa del ejemplo 2 pero escrito de tal manera que el bucle constituya un bloque.

```
10 REM Definición de la tabla de meses.
20 DIM M$(12)
30 FOR I = 1 TO 12 : READ M$(I) : NEXT I
40 DATA "ENE","FEB","MAR","ABR","MAY","JUN"
50 DATA "JUL","AGO","SEP","OCT","NOV","DEC"
60 REM Final de definición de la tabla.

100 INPUT "Iniciales del mes:",A$
110    REM Inicio de la búsqueda en una tabla.
120    LET I=1 : LET E=0
130    IF I>12 OR E THEN GO TO 200
140       LET E = A$ = M$(I)
150       IF NOT E THEN LET I = I + 1
160       GO TO 130
200    REM Inicio del bloque condicional
210    IF E THEN GO TO 250
220    PRINT "Iniciales erróneas"
230    GO TO 300
250    PRINT "Se trata del mes ";I
300    REM Fin del bloque condicional
310 GO TO 100
```

Ejemplo de estructura iterativa

Las líneas iniciales cumplen la misma misión que en el programa anterior. Debemos concentrar nuestra atención entre las líneas 110 y 300.

En una primera fase, desde la línea 110 hasta la 160, que es la que verdaderamente sustituye al FOR del programa anterior.

Como puede observar no se utiliza la instrucción FOR/NEXT sino que se ha sustituido por un esquema que incluye la instrucción IF.

La línea 120 inicia a 1 la variable *I* que es la equivalente a la variable de control del bucle en la pareja FOR/NEXT. Se inicia una variable *E* (de encontrado) que se inicia a 0, este valor se refiere al valor lógico NO.

La línea 130 es un IF que pregunta si *I* es menor que 12 o bien el valor de *E* ha alcanzado un valor verdadero. En cualquiera de los dos casos hay que abandonar el bucle. El primero, porque se ha sobrepasado el límite de la tabla sin encontrar una coincidencia con las iniciales del mes; en el segundo caso, porque se ha alcanzado coincidencia.

La línea 140 calcula el valor lógico de la variable *E*. Si hay coincidencia se coloca a un valor SI; si esta coincidencia no se da, se deja a NO, como está.

La línea 140 pregunta si esta coincidencia se ha alcanzado, para incrementar sólo en el caso de que no se haya alcanzado la variable I. De esta manera, siempre se deja el valor de I correcto, cuando se sale del bucle por coincidencia.

La línea 160 se recicla a la línea 130 para repetir la pregunta y según su resultado repetir el bucle o no.

Observe que en esta construcción las líneas 110 hasta la 150 constituyen un bloque según la definición ya que la única salida posible es a través de la línea 200.

Es verdad que se ha introducido una nueva variable (la *E*) y que el bucle se ha hecho más largo. En este ejemplo se puede admitir la primera forma, porque el programa no es muy largo. No obstante piense que, aunque los ejemplos deben ser necesariamente cortos, en programas largos las ventajas son importantes.

También debe tener en cuenta que no hay tanta diferencia con el FOR/NEXT. En este par de instrucciones también hay un IF escondido y un incremento de la variable *I*.

Para acabar de solucionar el problema se utilizan las instrucciones 200 hasta la 300 que constituyen un bloque condicional de alternativa simple. Según el valor de la variable *E* se imprime que las iniciales son erróneas y en caso contrario de qué número de mes se trata y que está almacenado en la variable *I*.

Finalmente, en la línea 310 se recicla a la línea 110 para entrar un nuevo valor de *A$*.

Una solución alternativa es escribir el bucle del modo siguiente:

```
110    REM Inicio de la búsqueda en una tabla.
120    LET I=0 : LET E=0
130    IF I>=12 OR E THEN GO TO 200
140       LET I = I+1
150       LET E = A$ = M$(I)
160       GO TO 130
```

que puede resultar en BASIC algo más eficiente. Observe que se inicia con la variable *I* a 0 y se incrementa a *I* al entrar en el bloque del bucle. También se ha modificado la pregunta. Cuando I ha alcanzado el valor 12 se debe abandonar el bucle ya que, si continuamos, se da a *I* el valor 13 que es incorrecto.

13.4.1 El bucle MIENTRAS

La construcción que hemos empleado para construir el bloque de bucle iterativo del programa anterior se denomina bucle MIENTRAS. La razón es porque puede describirse según las palabras:

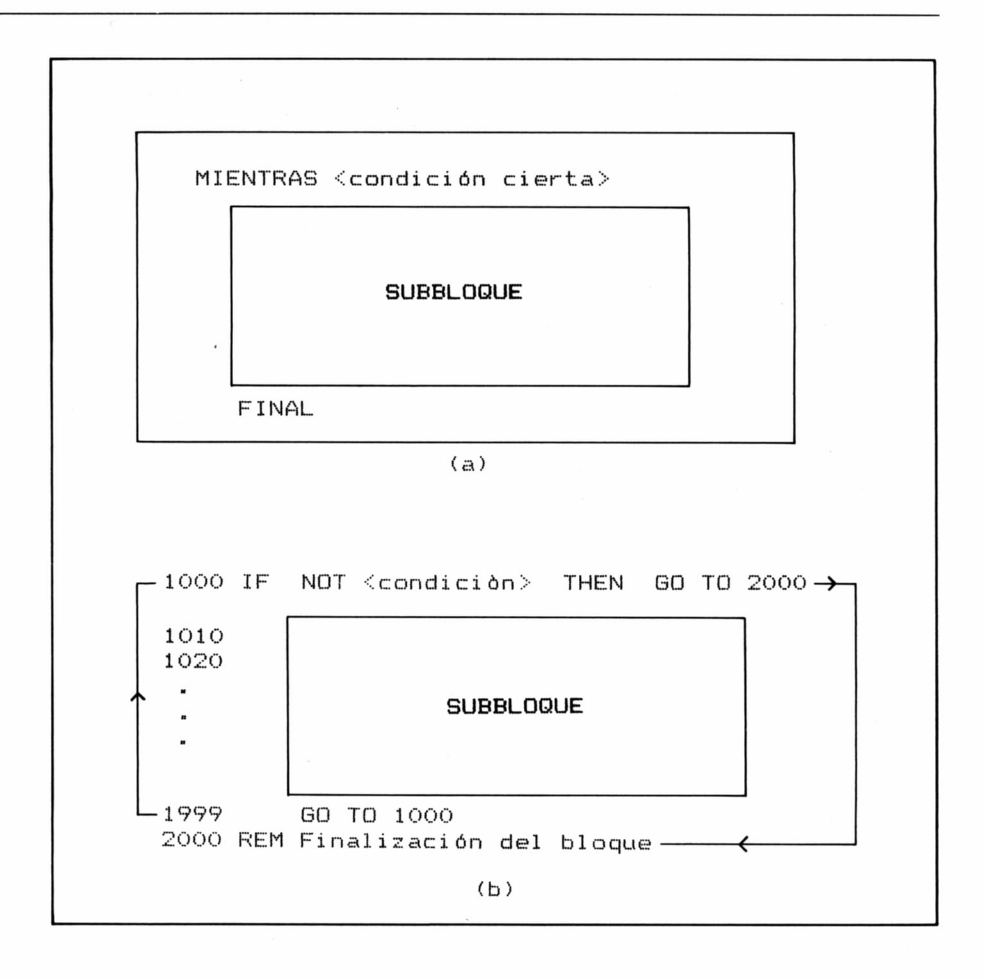
MIENTRAS No se sobrepase la tabla y no se encuentre

REALIZAR El incremento y la comparación

De una manera más formal la figura 10 muestra cuál es la manera de estructurarlo. En BASIC (parte b de la figura 10) se simula situando como primera instrucción de bloque el IF, que decide cuándo hay que repetir el bucle.

De hecho, en el programa anterior el bloque del bucle se inicia en la línea 130 si se quiere hacer rigurosamente. No en la línea 120 como se ha dicho antes, que constituye un bloque secuencial de dos instrucciones. Sin

Figura 10 a) Bloque MIENTRAS QUE; b) Simulación en BASIC del bucle MIENTRAS QUE



embargo, como la mayoría de veces se precisan estas inicializaciones se incluyen en el bloque del bucle.

La propiedad más importante del bucle construido de esta manera es que el subbloque se ejecuta cero o más veces. Es importante reconocer que según las condiciones el subbloque interno del bucle no se ejecuta.

En el ejemplo que hemos tratado, el bloque se ejecuta siempre una vez al menos. Observe que la condición siempre se cumple la primera vez ya que inmediatamente antes se ha colocado la variable *I* a 1 y la variable *E* a 0. Este es un mal ejemplo. Aunque se escribe como un bucle MIENTRAS, forzamos a que no cumpla la propiedad genuina de que en algunas ocasiones no se ejecute el bloque interno.

Vamos a considerar otro ejemplo que sí cumpla con las propiedades estrictas de un bloque MIENTRAS.

Se trata de realizar un programa que lee una variable textual y que pretende hallar las palabras que contiene. El concepto de palabra es muy amplio; es cualquier secuencia de símbolos que no contenga el blanco. Las palabras se separan por el símbolo espacio o blanco.

El programa es el siguiente:

```
  10 REM Entrada del texto
  20 INPUT "Entre el texto:";A$
1000 REM Contar palabras
1010 LET E=0 : LET P = 0 : LET I = 1
2000 IF I > LEN(A$) THEN GO TO 3000

2500    LET C$ = MID$(A$,I,1)
2510    IF E = 1 THEN  GO TO 2700

2600        IF C$ <> " " THEN LET E = 1 : LET P = P+1
2610        GO TO 2900

2700        IF C$ =  " " THEN LET E = 0

2900     LET I = I +1
2910     GO TO 2000

3000 PRINT "Hay ";P;" palabras"
3010 GO TO 10
```

El método consiste en analizar cada uno de los caracteres reconociendo los blancos para separar las palabras.

La línea 10 sirve para entrar el texto a analizar.

La línea 1010 contiene tres inicializaciones de las variables, la variable *I* cuenta el carácter que se analiza en la vuelta del bucle. La variable *P* cuenta las palabras que hemos encontrado.

La variable *E* es una variable lógica que nos indica si estamos dentro de una palabra o no. Tenga en cuenta que las palabras pueden estar separadas por más de un blanco.

En el momento que hemos empezado a analizar una palabra hay que pasar caracteres hasta encontrar un blanco. En este momento debemos cambiar al estado «fuera de palabra» y analizar caracteres hasta encontrar uno distinto de blanco, entonces hay que colocar el estado «en palabra» y contabilizar una nueva palabra.

La situación inicial es la de colocarse en «fuera de palabra», si los caracteres iniciales son blancos se alcanza el primer símbolo distinto de blanco cambiando correctamente. Se indica que se está en la palabra cuando el valor de *E* es 1 y con 0 el caso contrario.

La línea 2000 inicia el bucle MIENTRAS. Desde la 2500 hasta la 2900 es el subbloque en que se descompone la estructura iterativa y la línea 3000 es la finalización del bloque. Se han dejado separaciones claras para subrayar cuáles son los bloques.

El subbloque es a su vez un bloque condicional, se pregunta si el estado es «en palabra». En caso afirmativo se bifurca el flujo del programa a la línea 2700. En caso contrario, es decir, estamos «fuera de palabra», se alcanza la línea 2600, que pregunta si el carácter a analizar es distinto de blanco. En caso afirmativo, se coloca el estado «en palabra» y se cuenta una palabra más. Si el carácter es blanco se va a analizar el siguiente.

Cuando se alcanza la línea 2700, es que el estado es «en palabra». Se pregunta si el carácter es blanco. En caso afirmativo se cambia al estado «fuera de palabra», con los demás caracteres se alcanza la línea 2900.

En la línea 2900 se incrementa el valor de *I* para analizar el carácter siguiente y se inicia el bucle otra vez.

Esto ocurre mientras el valor de *I* no supere la longitud del texto.

Finalmente la línea 3000 imprime el resultado y la línea 3010 envía el control a leer otro texto.

Ha de notar que si introduce el texto vacío el bucle no se ejecuta ninguna vez, propiedad típica del bucle MIENTRAS.

En este ejemplo se ha de señalar que el proceso es más repetitivo que iterativo, pues en este caso, sabemos de antemano cuántos caracteres hay que analizar. De hecho utilizamos la función LEN que los cuenta cada vuelta. En otros lenguajes es posible recorrer una cadena de caracteres y obtener una señal cuando se alcanza el último carácter. En este caso la cuenta previa no se realiza.

En BASIC, la utilización de un FOR para un caso de este tipo no es aconsejable pues muchos de los dialectos recorren una vez el interior del bucle cuando el valor final es menor que el valor inicial. Posiblemente con un FOR el bucle es más eficiente pues la función LEN sólo se evalúa una vez.

El programa escrito con la instrucción FOR es el siguiente:

```
  10 REM Entrada del texto
  20 INPUT "Entre el texto:",A$
1000 REM Contar palabras
1010 LET E=0 : LET P = 0
2000 IF A$="" THEN GO TO 3000
2010 FOR I = 1 TO LEN(A$)

2500     LET C$ = MID$(A$,I,1)
2510     IF E = 1 THEN  GO TO 2700

2600          IF C$ <> " " THEN LET E = 1 : LET P = P+1
2610          GO TO 2910

2700          IF C$ =  " " THEN LET E = 0

2910      NEXT I

3000 PRINT "Hay ";P;" palabras"
3010 GO TO 10
```

Como puede observar la única precaución que hay que observar es colocar el IF de la línea 2000 para considerar el caso de que la longitud del texto es nula. Esta precaución es innecesaria en los dialectos del BASIC en los que el FOR no se ejecuta ninguna vez si el valor final es menor que el valor inicial.

Vea cómo el subbloque del interior del bucle es más complicado que en el ejemplo de la búsqueda pero se entiende el texto con facilidad debido a esta partición en subbloques.

Iteración para hallar raíces cuadradas

Finalmente considere el siguiente ejemplo de construcción realmente iterativa. Se trata de hallar la raíz cuadrada de un número, naturalmente sin utilizar la función que el BASIC proporciona. Se trata de un ejemplo académico para que entienda el concepto puro de proceso iterativo.

Para encontrar la raíz cuadrada de un número *n* hay que encontrar un número *x* que multiplicado por si mismo dé *n*. En los ordenadores no se utiliza la forma manual de hallar la raíz cuadrada sino que se realiza por el método siguiente.

Para hallar la raíz cuadrada del número 16 (ya sabemos que el resultado es 4) se toma y se divide por 2, con lo que se obtiene un 8, ahora 16 se puede expresar como el cociente por el divisor (8×2) más el resto, en este caso cero. Si conseguimos que cociente y divisor sean iguales y el resto cero ya hemos encontrado la raíz cuadrada. Para conseguir que el resto sea cero basta utilizar tantos decimales como queramos.

Probamos entonces con el valor promedio de 8 y 2 que es 5, obtenemos un cociente que es 3,2, como puede ver 5 y 3,2 sólo se diferencian en 2,2 mientras en el primer caso, se diferenciaban 6 (8 y 2). El proceso se repite hasta encontrar la raíz cuadrada.

La tabla siguiente muestra el proceso

Divisor	**Cociente**	**Media**
2	8	5
5	3,2	4,1
4,1	3,9024	4,0012
4,0012	3,9988	4,0
4,0	4,0	4,0

Después de unas cuantas veces llegamos al valor de 4 que es la raíz cuadrada de 16.

El programa que realiza este proceso es el siguiente:

```
 10 INPUT "Número a sacar la raíz cuadrada:",N
 20 IF N<=0 THEN END
 30   INPUT "Número inicial de prueba:",D
 35   PRINT "Divisor";TAB(10);"Cociente";TAB(20);"Media"
 40   LET C = N/D
 45   PRINT D;TAB(10);C;
 50   IF C = D THEN GO TO 100
 60       LET D = (C+D)/2
 65       PRINT TAB(20);D
 70       LET C= N/D
 75       PRINT D;TAB(10);C;
 80       GO TO 50
100    PRINT :PRINT "La raíz cuadrada es :";C
110    GO TO 10
```

En la línea 10 se pide el número del cual se desea sacar la raíz cuadrada. Si este número es cero o negativo se termina el programa porque no tiene sentido las raíces cuadradas de número negativos (de hecho constituyen una clase de resultados llamados números imaginarios).

En la línea 30 se pide el número por el que se desea empezar. El número de vueltas que se realizan depende fuertemente de este número. Así

si para sacar la raíz cuadrada de 16 utilizamos el 4 como divisor inicial no se da ninguna vuelta.

Después de este divisor inicial se calcula el cociente en la línea 40.

La línea 50 pregunta si el cociente obtenido y el divisor son iguales. En caso afirmativo se envía a escribir el resultado en la línea 100.

Si no son iguales se saca la media (línea 60) y con este divisor se vuelve a sacar el cociente (línea 70). El programa se recicla (línea 80) a la línea 50 para averiguar si el cociente y el divisor son iguales.

La línea 110 envía el programa para preguntar un nuevo número.

Las líneas acabadas en 5 (35, 45, 65 y 75) se colocan para que el programa realice la tabla del proceso según se ha indicado más arriba.

Para probar el programa, calcule la raíz cuadrada de 16 con los valores iniciales de 2. Debe reproducir la tabla de arriba. Y con el valor de 4 en el que no se debe dar ninguna vuelta.

Si prueba de calcular la raíz de 2 o de 3, es posible que el programa no salga jamás del bucle (si no es mediante la tecla de interrupción). La razón es que no puede darse en este caso la igualdad absoluta de los decimales, ya que la raíz cuadrada de estos números pertenece a una clase que se denominan números irracionales. Sólo es posible expresarla exactamente con infinitos decimales. Por otra parte, es posible que le suceda con otros números ya que la máquina posee un número finito de decimales.

El problema de precisión

Para evitar este problema es mucho más adecuado cambiar la instrucción 50 por la siguiente:

```
50 IF ABS(C-D) < 0.00001 THEN GO TO 100
```

en lugar de preguntar por la igualdad absoluta, se pregunta cuando coinciden con tantos decimales (el 0.00001 significa que coinciden con 4 decimales como mínimo) como especifiquemos. Naturalmente se saca el valor absoluto de la diferencia pues nos importa poco que el cociente o el divisor sea uno mayor que otro.

Este tipo de preguntas son muy frecuentes en cálculo numérico con números decimales ya que debido a la precisión finita de la máquina no es posible encontrar la igualdad absoluta.

Como puede Ud. comprobar se trata de un verdadero proceso iterativo, pues cumple con las dos propiedades.

a) No se sabe con certeza el número de vueltas que se realizarán. Depende incluso del valor inicial que demos.

b) Puede ser que no se realice ninguna vuelta. Como en el caso de calcular la raíz de 16 y entrar como valor inicial 4.

En conclusión, en los procesos iterativos es recomendable utilizar la composición en bloques con instrucciones IF antes que salir del interior de un bucle construido con el par FOR/NEXT que indica un proceso repetitivo.

13.4.2 El bucle HASTA

Volvamos al ejemplo de búsqueda en una tabla de las iniciales del mes. Un método alternativo para escribir el programa es el siguiente:

```
 10 REM Definición de la tabla de meses.
 20 DIM M$(12)
 30 FOR I = 1 TO 12 : READ M$(I) : NEXT I
 40 DATA "ENE","FEB","MAR","ABR","MAY","JUN"
 50 DATA "JUL","AGO","SEP","OCT","NOV","DEC"
 60 REM Final de definición de la tabla.

100 INPUT "Iniciales del mes:",A$
110    REM Inicio de la búsqueda en una tabla.
120    LET I=0 : LET E=0
130       LET I = I+1
140       LET E = A$ = M$(I)
160       IF I<12 AND NOT E THEN GO TO 130
200    REM Inicio del bloque condicional
210    IF E THEN GO TO 250
220    PRINT "Iniciales erróneas"
230    GO TO 300
250    PRINT "Se trata del mes ";I
300    REM Fin del bloque condicional
310 GO TO 100
```

La estructura global es la misma que en el programa anterior, por lo tanto, sirven los mismos comentarios excepto en el bloque de búsqueda, es decir, de las instrucciones 120 hasta 150.

La diferencia más importante es que se inicializa la variable *I* a 0 y se inicia el bloque iterativo en la línea 130 con el incremento de la variable *I*. A continuación se calcula la variable *E*, que es SI si coinciden las iniciales y es NO si no coinciden.

En la línea 160 se pregunta si no se ha alcanzado el final de la tabla y aún no se ha alcanzado la coincidencia. En caso afirmativo se recicla a la línea 130. En caso negativo se abandona el bucle.

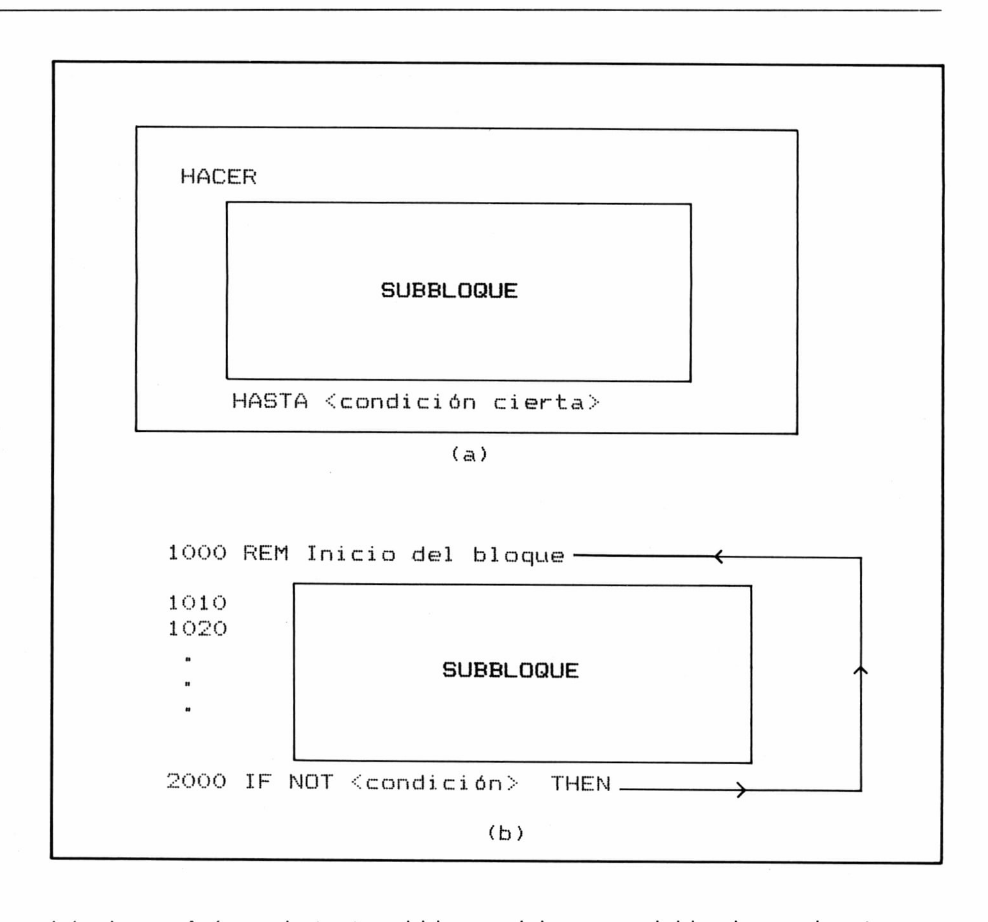
Esta construcción del bucle es más natural para el problema. El bucle debe ejecutarse al menos una vez y la situación de la pregunta al final del bloque es más conveniente.

Este tipo de bucle se conoce como el bucle HASTA. La figura 11 muestra el esquema de funcionamiento y su simulación en BASIC. El esquema verbal de la construcción es:

HACER El incremento y la comparación
    HASTA que se sobrepase la tabla o se encuentre.

La característica más importante de este tipo de bucle es que la pregunta que decide si continuar o no la repetición o iteración se sitúa al final

Figura 11 a) Bloque HASTA QUE; b) Simulación en BASIC del bucle HASTA QUE



del mismo. Así, por lo tanto, el bloque del cuerpo del bucle se ejecuta por lo menos una vez.

La otra característica es que se utiliza en procesos iterativos en los que se desconocen de antemano el número de vueltas que hay que dar.

Relaciones entre los bloques MIENTRAS y HASTA

Este tipo de bucle para procedimientos iterativos es muy parecido al de MIENTRAS. La diferencia es que la decisión se sitúa al final y, por lo tanto, el bloque interior del bucle se ejecuta una vez.

Es fácil notar una característica importante en las construcciones MIENTRAS y HASTA que las relacionan. Un bucle HACER... HASTA se puede simular con un bucle MIENTRAS.

Esto se ha realizado precisamente en el apartado anterior en la búsqueda del mes en una tabla. La inicialización de las variables fuerza que el interior del bucle se ejecute por lo menos una vez. Al colocar la pregunta al inicio del bucle podemos comenzar con un valor de las variables que necesariamente obliguen a entrar en el bucle. La versión del programa de la búsqueda de las iniciales del mes con un bucle MIENTRAS utiliza esta particularidad.

Por otra parte, la inversa no es cierta. Con un bucle HACER... HASTA no es posible simular un bucle MIENTRAS, pues no hay manera de evitar que por lo menos se ejecute una vez.

Un ejemplo de esta construcción está en los bucles de órdenes de los programas, en general. Cualquier programa necesita recibir órdenes para ejecutar las operaciones, como en el caso del editor de la tabla.

La estructura es iterativa, pues no sabemos nunca el número de órdenes que vamos a ejecutar, y por lo menos hay que ejecutar una instrucción, la de pedir qué operación deseamos.

El esquema general es:

HACER

Entrar una orden.

Bloque condicional que reparte

las operaciones según la orden.

HASTA que la operación sea finalizar

13.4.3 El bucle generalizado

En los dos apartados anteriores se han estudiado las estructuras iterativas que tienen como principal propiedad la de que no se sabe cuántas veces se ejecutan los bloques interiores de estas estructuras.

Hemos estudiado los dos casos extremos que corresponden a decidir cuándo hay que abandonar este bucle interior. Se toma la decisión al empezar el bloque (estructura MIENTRAS) o al finalizar (estructura HASTA).

¿Es posible pensar en una situación intermedia? Es decir, se puede colocar la pregunta en un punto intermedio. La respuesta es sí pero hay que matizarla.

La programación estructurada indica que sólo debe haber una pregunta; es decir, sólo se alcanza el final de esta estructura desde un punto. Pero que este punto puede situarse en cualquier lugar del bucle, o mejor se utilizan dos subbloques interiores.

La representación de esta estructura se recoge en la figura 12. Aparecen dos subbloques que hemos denominado A y B. La estructura se inicia con la ejecución del subbloque A. Se pregunta a continuación por una condición. Si ésta es cierta, se ejecuta el subbloque B, si es falsa se sale del bucle. Una vez finalizado el subbloque B se inicia otra vez el subbloque A.

El subbloque A se ejecuta siempre al menos una vez. El subbloque B puede no ejecutarse ninguna vez.

Es fácil ver que la estructura MIENTRAS se deriva desde esta construcción cuando el subbloque A no contiene ninguna instrucción. La estructura HASTA se obtiene cuando el subbloque B no contiene ninguna instrucción.

Propiedad del bucle generalizado

Las propiedades de este bucle generalizado son:

a) Es iterativo. Se puede realizar sin saber el número de veces que se repite de antemano.

b) La instrucción de salida se sitúa en una instrucción únicamente.

c) Los subbloques pueden ser vacíos, es decir, sin ninguna instrucción.

d) El subbloque A se ejecuta una vez más que el subbloque B.

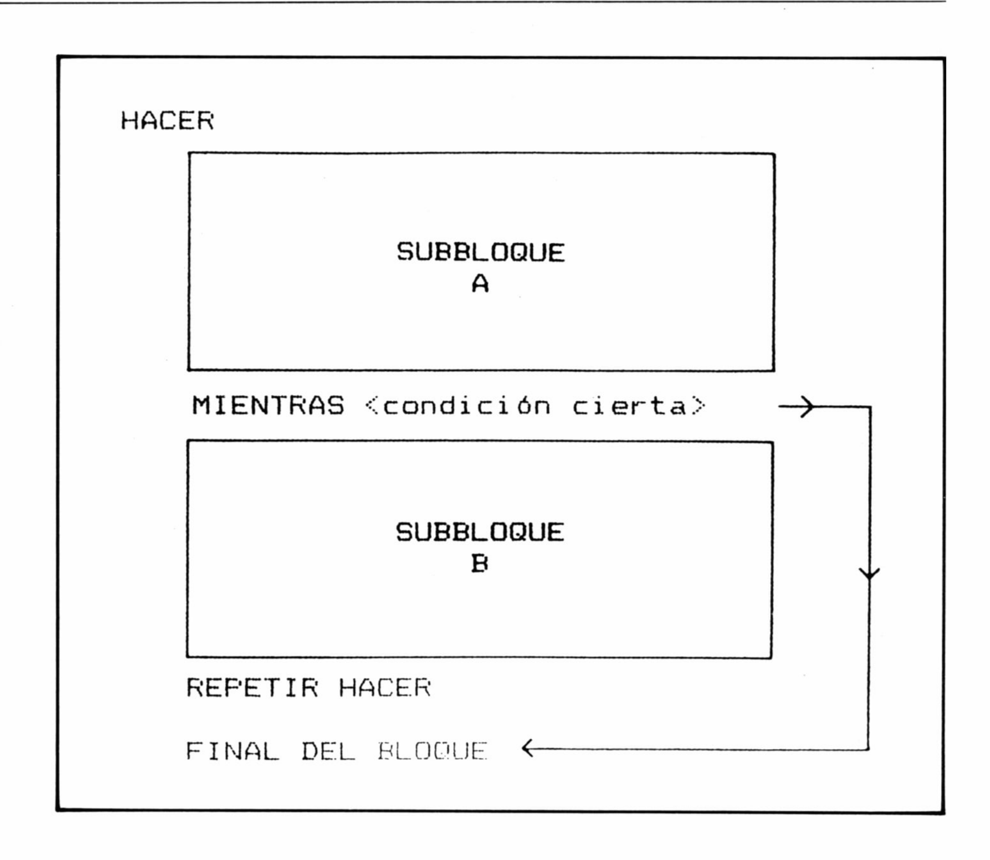


Figura 12 Bloque iterativo general

e) El subbloque A se ejecuta al menos una vez.

f) El subbloque B puede no ejecutarse.

La figura 13 muestra la simulación en BASIC de esta estructura.

Dentro de la disciplina de la programación estructurada este tipo de estructura iterativa es la que mejor se adapta al lenguaje BASIC. En este lenguaje no hay instrucciones específicas para conseguir la estructuración del programa en bloques a diferencia de lo que ocurre en otros lenguajes.

Es necesario que tenga en cuenta que esta generalización no le permite utilizar la instrucción FOR con un IF de salida porque se puede salir también por el final del FOR.

Considere de nuevo la estructura del programa de la búsqueda de meses con la instrucción FOR, tal como escribimos a continuación

```
110 FOR I = 1 TO 12
120    IF A$=M$(I) THEN GO TO 200
130    NEXT I
```

tiene dos salidas una por la línea 200 y otra por la línea siguiente a la 130. Esto no se permite en la construcción de un bucle generalizado, pues siempre la programación estructurada trabaja con bloques que tienen una sola entrada y una sola salida.

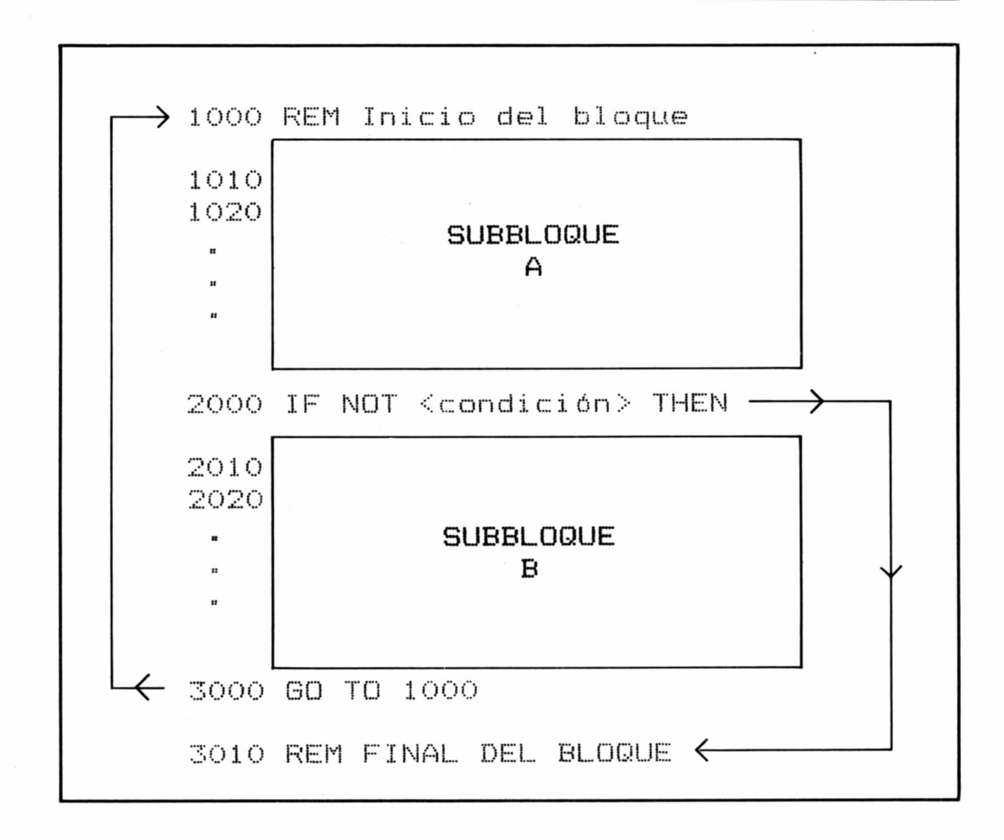


Figura 13 Simulación en BASIC de un bucle general

Un ejemplo de bucle generalizado es el que hemos puesto como bucle de órdenes en el apartado anterior:

HACER

Entrar una orden.

MIENTRAS orden no sea final.
Bloque condicional que reparte
las operaciones según la orden.

REPETIR HACER

De hecho este esquema se ha repetido muchas veces en esta Enciclopedia, cuando se repite un mismo programa.
La estructura de

HACER

Entrar el primer dato.

MIENTRAS este primer dato tiene sentido.
Ejecutar el resto del programa.

REPETIR EL PROGRAMA.

se ha repetido hasta la saciedad.

Como ejemplo concreto tome el programa de las raíces cuadradas. La estructura de bucle generalizado es la siguiente:

La instrucción 10 es el subbloque A.

La instrucción 20 es la pregunta, si el número es mayor que 0.

Ejecutar el resto de instrucciones.

GO TO a la línea 10.

13.5 LA PROGRAMACION ESTRUCTURADA Y LOS ORDINOGRAMAS

En el capítulo 7, en el tomo II, se han visto técnicas de expresión gráficas de programas mediante los ordinogramas.

Algunos autores han dicho que los ordinogramas son incompatibles con la programación estructurada. A nuestro juicio es una afirmación exagerada ya que la programación estructurada es una disciplina y como tal puede utilizarse perfectamente con los ordinogramas.

La disciplina de la programación estructurada únicamente impone restricciones en la confección de bloques y en sus relaciones. Cuando se quiere representar algún problema de modo gráfico la mejor técnica es utilizar los ordinogramas con la misma filosofía que la de los bloques, es decir, cada bloque de programación se desarrolla mediante un ordinograma.

Según el tipo de bloques es posible definir en un mismo dibujo uno o dos niveles de bloques, pero se debe ser reacio a colocar más niveles dentro de un ordinograma.

De hecho cuando hablamos de ordinograma ya decíamos que el tamaño adecuado era de una página de papel.

Los subbloques se expresan en los ordinogramas como rectángulos de proceso

La definición de los subbloques se hace mediante los rectángulos típicos de procesos en los ordinogramas.

A manera de conclusión, con este capítulo se cierra el verdadero núcleo de lo que es programación. En ella se han dado los cimientos no sólo de las instrucciones en BASIC sino que también hemos procurado que adquiera un estilo correcto.

Recordando lo que se decía en la introducción, en estos momentos ha aprendido a leer programas escritos por otros, a escribir algunos sencillos. Ahora le queda el camino de escribir cada vez más complejos para que esto que ha aprendido tenga la máxima utilidad.

A partir de ahora se toman temas especiales, pero importantes como son los gráficos y una introducción a los ficheros.

A pesar de haber adquirido una técnica, deseamos recordarle que el problema más importante es comprender bien el problema a resolver. Muchos programadores nunca llegan a escribir programas del todo correctos porque no dedican suficiente atención a un problema.

Entre los programadores se dice que la corrección del programa, entendiendo como corrección, que el programa haga lo que se desea que haga, es inversamente proporcional al tiempo que se tarda a teclearlo.

Con esto se quiere decir que antes de situarnos delante del teclado del ordenador hay que gastar el tiempo suficiente en reflexionar sobre el problema para que cuando se escriba realmente sobre el ordenador todos los detalles estén pensados.

RESUMEN

Las estructuras de repetición permiten la ejecución de un bloque más de una vez.

La característica principal es que sabemos el número de veces que hay que repetir el bloque de iniciar el bucle.

La estructura FOR/NEXT del BASIC está bien adaptada a los esquemas repetitivos.

Para que sea realmente una estructura repetitiva es necesario que no haya instrucción de salto fuera del bucle.

La estructura iterativa es la que permite la ejecución del bloque más de una vez, pero desconocemos de antemano el número de veces que se repite el subbloque interior.

La instrucción FOR/NEXT no sirve para las estructuras iterativas porque es necesario salir del interior del bucle mediante un salto. Por lo tanto, este tipo de construcción tiene dos salidas en oposición a la regla fundamental de la programación estructurada.

La estructura iterativa consiste, de una manera general, en dos subbloques interiores denominados A y B separados por una pregunta.

El funcionamiento consiste en acceder al subbloque A. Una vez terminado se formula la pregunta, si es afirmativa se continúa por el subbloque B. Una vez finalizado el subbloque B se recicla el bucle hacia el subbloque A.

Si la pregunta es negativa se sale de la estructura.

Las propiedades de esta estructura son:

a) Es iterativa.

b) La pregunta se sitúa en único punto.

c) Los subbloques A o B pueden ser vacíos.

d) El subbloque A se ejecuta una vez más que el subbloque B.

e) El subbloque A se ejecuta al menos una vez.

f) El subbloque B puede no ejecutarse.

Cuando el subbloque A es vacío la estructura se denomina de MIENTRAS. El único subbloque que contiene puede no ejecutarse.

Cuando el subbloque B es vacío la estructura se denomina HASTA. El único subbloque se ejecuta al menos una vez.

Los ordinogramas se pueden utilizar para representar gráficamente las estructuras de la programación estructurada.

No deben utilizarse más de dos niveles de estructuración en un mismo ordinograma, que a su vez no debe ocupar más de una página aproximadamente.

Los subbloques que quedan por definir en un ordinograma se simbolizan por un rectángulo de proceso que contiene la referencia del nombre del subbloque que se desarrolla en un ordinograma aparte.

EJERCICIOS DE AUTOCOMPROBACION

Completa las frases siguientes:

21. Cuando hay que repetir un bloque se utiliza una estructura ...... o ......

22. En la estructura repetitiva se ...... el número de veces que se repetirá el bloque antes de empezar

23. En la estructura ...... no se sabe el número de veces que se repetirá el bloque antes de empezar.

24. El par de instrucciones FOR/NEXT se adaptan bien a la estructura ......

25. El bucle generalizado contiene ...... subbloques.

26. El subbloque A se ejecuta una vez más que el ...... en una estructura iterativa generalizada.

27. Cuando el subbloque A es vacío la estructura se denomina ......

28. Cuando el subbloque B es vacío la estructura se denomina ......

29. Se puede programar estructuradamente y utilizar los ......

30. Los subbloques se simbolizan como ...... en los ordinogramas, que contienen el nombre del proceso.

Encierre en un círculo la letra que corresponda a la alternativa correcta.

31. Un programa debe escribir 5 páginas de texto. La estructura que precisa para estructurar esta tarea es:

    a) Secuencial.

    b) Condicional.

    c) Iterativa.

    d) Repetitiva.

32. Una estructura MIENTRAS es:

    a) Secuencial.

    b) Condicional.

    c) Iterativa.

    d) Repetitiva.

33. ¿En qué estructuras de las siguientes debe ejecutarse un bloque una vez por lo menos?

    a) Condicional alternativa simple.

    b) HASTA.

    c) MIENTRAS.

    d) Condicional alternativa múltiple.

34. Un programa debe recibir órdenes del teclado de un ordenador ¿qué estructura es más conveniente?

    a) Secuencial.

    b) Condicional.

    c) Iterativa.

    d) Repetitiva.

35. Cuántos niveles de subbloques es recomendable utilizar en los ordinogramas.

    a) Ninguno.

    b) Uno.

    c) Dos.

    d) Tres.

36. En el esquema de programa siguiente:

    Leer una línea.

    Mientras no sea la última leer la siguiente.

    Si hay líneas
        Ordenarlas por orden alfabético.
        Escribirlas en la pantalla.

    en caso contrario Escribir No hay líneas.

    ¿qué estructura de las siguientes no interviene?

    a) Bucle generalizado.

    b) Secuencial.

    c) Condicional.

    d) Bucle MIENTRAS

37. ¿Cuál de los programas siguientes no cumple con las reglas de la programación estructurada?

a)

```
10 FOR I = 1 TO 10
20 PRINT "TERMINO ";I
30 NEXT I
```

b)

```
10 FOR I = 1 TO 10
20 IF I<5 THEN PRINT "TERMINO ";I
30 NEXT I
```

c)

```
 10 FOR I = 1 TO 10
 20 IF I= INT(I/2)*2 THEN GO TO 100
 30 PRINT "TERMINO ";I
 40 NEXT I
 50 END
100 PRINT "TERMINO PAR ";I
110 GO TO 10
```

d)

```
10 FOR I = 1 TO 10
20 IF I= INT(I/2)*2 THEN GO TO 50
30 PRINT "TERMINO IMPAR ";I
40 GO TO 80
50 PRINT "TERMINO PAR ";I
80 NEXT I
```

38. En el programa del apartado 13.4.1 del texto que sirve para contar palabras.
¿Qué instrucciones abarca el bucle MIENTRAS?

a) 1000-3000

b) 2000-2910

c) 1000-2910

d) 1000-1010

39. En el programa del apartado 13.4.1 del texto que sirve para contar palabras.
¿Qué instrucciones abarca el bloque condicional para saber en que estado se está, «en palabra» o «fuera de palabra»?

a) 2510-2700

b) 2510-2610

c) 2600-2700

d) 2500-2900

40. En el programa para sacar raíces cuadradas se puede utilizar un bucle generalizado. ¿Cuál de los programas siguientes realiza esta posibilidad? (sólo se ponen las instrucciones del bucle propiamente)

a)

```
40 LET C = N/D
50 IF C=D THEN GO TO 100
60 LET D = (C+D)/2
70 GO TO 40
```

b)

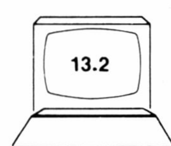
```
40 LET C = N/D
50 IF C=D THEN GO TO 100
60 LET D = (C+D)/2
70 GO TO 50
```

c)

```
40 LET C = N/D
50 IF C=D THEN GO TO 100
60 LET D = (C+D)/2
70 LET C = N/D
80 GO TO 40
```

d)

```
40 LET C = N/D
50 IF C=D THEN GO TO 40
60 LET D = (C+D)/2
70 GO TO 40
```



◄

Capítulo 14
• Gráficos por ordenador

Objetivos	
Introducción	
Caracteres gráficos	
Gráficos de alta resolución	Ejes de coordenadas Operaciones primitivas Pendiente de las líneas rectas Operaciones auxiliares Trigonometría
Gráficos bidimensionales	Trazado de polígonos y circunferencias Elipses y espirales Figuras de Lissajous
Diagramas	Diagramas de barras Diagramas circulares Diagramas X-Y
Superposiciones de líneas	Borrado de líneas Dibujo interactivo

14.0 OBJETIVOS

Los gráficos por ordenador han pasado de ser un dominio reservado para ingenieros y diseñadores a ser una característica habitual de los ordenadores personales. En este capítulo aprenderemos los conceptos básicos de los gráficos por ordenador.

En primer lugar veremos cómo construir tablas y dibujos utilizando el juego de caracteres gráficos. Aquí veremos la primera asociación entre el contenido de la memoria y lo que está representado en la pantalla. En segundo lugar, aprenderemos los conceptos fundamentales de los gráficos de alta resolución. Relacionaremos la calidad de la pantalla con el concepto de resolución y veremos la definición de «pixel» que constituye la base en que se fundamenta este tipo de gráficos. Estudiaremos también las operaciones primitivas utilizadas para construir los gráficos. Al mismo tiempo, realizaremos un breve repaso de los conceptos matemáticos asociados a este tema.

Finalmente, aplicaremos estos conceptos a la realización de casos prácticos como son los gráficos bidimensionales (polígonos, círculos, elipses...), trazado de diagramas (de barras, circulares y X-Y) y un programa de dibujo interactivo.

14.1 INTRODUCCION

Las instrucciones para realizar gráficos no forman parte del BASIC estándar. No obstante, debido a su indudable atractivo, la mayoría de ordenadores incorporan instrucciones para realizar algún tipo de gráficos. El empleo de gráficos por ordenador tiene interés por varios motivos.

La utilización de gráficos facilita el aprendizaje de programación

El primero y más evidente es la presentación de resultados. Un conjunto de resultados es mucho más fácil de entender si se representa en forma de gráfico que si se imprime una simple lista de números.

El segundo motivo está relacionado con el aprendizaje de la programación. Cuando en un programa corriente realizamos unas operaciones, los pasos intermedios no son visibles hasta que no damos una orden PRINT para visualizar el resultado final. Por el contrario, en un programa que realice gráficos, casi todos los pasos tienen su reflejo en la pantalla. De esta forma es muy fácil apreciar dónde y por qué se ha producido un error y, lo que es más importante, sabremos fácilmente cómo corregirlo.

El tercer motivo es de índole didáctica. Para realizar gráficos se requiere tener unos conocimientos de Geometría, Trigonometría, representación en ejes cartesianos, etc. En general, esta parte de las Matemáticas es de difícil aprendizaje ya que incluye muchas fórmulas y conceptos, a menudo complicados. Sin embargo, la utilización conjunta de fórmulas y su correspondiente representación visual hace que los conceptos matemáticos aparezcan claros y fáciles de entender.

Existen dos procedimientos para realizar gráficos por ordenador. El primero se basa en la utilización de caracteres gráficos y el segundo se basa en considerar la pantalla como un mosaico de puntos (gráficos de alta resolución).

En esta primera parte utilizaremos únicamente los gráficos en blanco y negro.



14.2 CARACTERES GRAFICOS

Aparte de los caracteres normales incluidos en el código ASCII, la mayoría de ordenadores incluyen un conjunto de caracteres gráficos. Estos caracteres tendrán códigos superiores a 127.

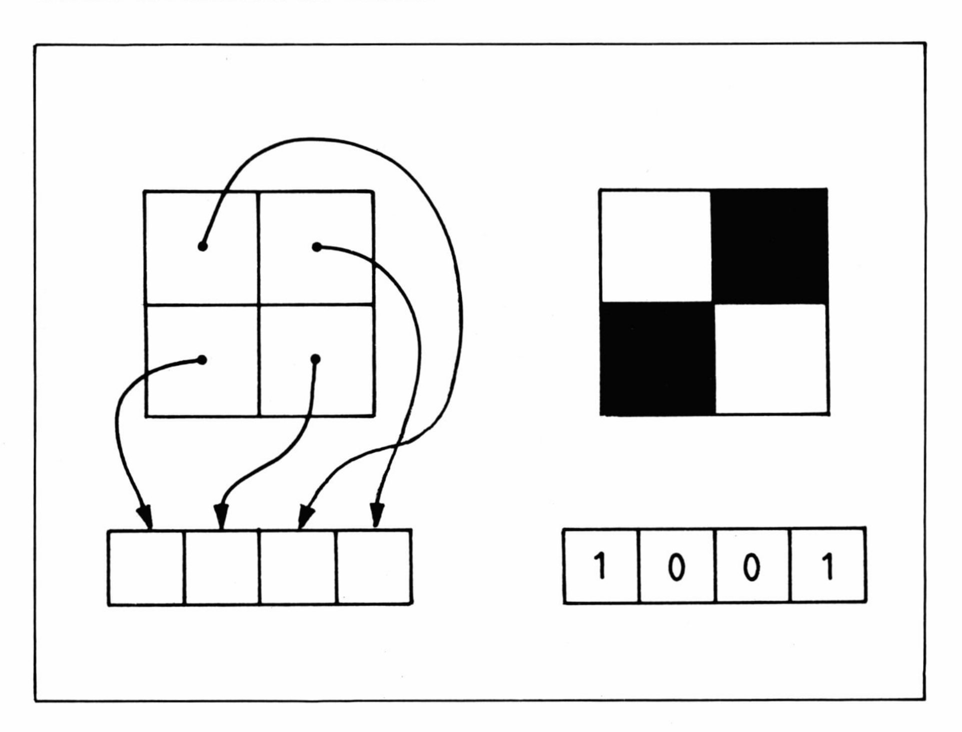
Si observamos la representación en pantalla de los caracteres normales, veremos que todos ellos ocupan siempre la misma anchura, tanto si se escribe una letra ancha como la M o el símbolo del punto (.). En realidad, todos los caracteres se encuentran inscritos dentro de un recuadro que tiene siempre las mismas dimensiones. No existe, además, separación entre los recuadros, y si colocamos recuadros adyacentes en todas direcciones ocuparemos toda la pantalla sin dejar ningún resquicio.

Los caracteres gráficos tienen el mismo tamaño que los normales

Los caracteres gráficos también ocupan un recuadro. Veremos seguidamente cómo se forman estos caracteres. Cada recuadro se encuentra dividido en 4 o 6 partes según el ordenador que empleemos.

Para seguir con nuestra explicación supondremos que son cuatro partes (Fig. 1). Cada una de estas partes se encuentra asociada a un bit. Este bit pertenece al byte que contiene el código del carácter. Si un bit es cero, entonces la zona asociada está en blanco. Si el bit vale 1, entonces la zona está en negro. Las combinaciones de cuatro bits nos darán los 16 caracteres posibles. (Si el recuadro se divide en 6 partes entonces hay 64 caracteres.) Estos caracteres se obtienen mediante la función CHR$ con argumentos mayores que 127. En algunos ordenadores también se pueden obtener directamente del teclado.

Figura 1 Relación entre un carácter gráfico y un grupo de bits. Ejemplo de un carácter construido según el valor 1001.



Los caracteres gráficos ocupan códigos distintos de los ASCII

Como ejemplo de utilización, dibujaremos un cuadrado. Para ello emplearemos los caracteres de la figura 2 para construir los vértices y los caracteres de la figura 3 para las líneas horizontales y verticales.

Programa de «Dibujo cuadrado»

```
 10 REM Dibujo cuadrado
 20 PRINT CHR$(139);
 30 FOR I=2 TO 11: PRINT CHR$(131); : NEXT I
 40 PRINT CHR$(135);
 50 FOR I=2 TO 11
 60   PRINT AT(1,I);CHR$(138);AT(12,I);CHR$(133);
 70   NEXT I
 80 PRINT AT(1,12);CHR$(142);
 90 FOR I=2 TO 11: PRINT CHR$(140); : NEXT I
100 PRINT CHR$(141);
```

La línea 20 dibuja el vértice superior izquierdo. La línea 30 es un bucle FOR/NEXT completo que escribe la línea horizontal. Obsérvese el punto y coma (;) al final de la instrucción PRINT para escribir seguido. La línea 40 escribe el vértice superior derecho. Las líneas 50, 60 y 70 escriben simultáneamente las dos líneas verticales. La línea 80 escribe el vértice inferior izquierdo. El bucle FOR/NEXT de la línea 90 escribe la raya horizontal inferior y finalmente la línea 100 dibuja el vértice inferior derecho.

Cuando hay que separar diversas columnas de números, que forman parte de una tabla, se suelen emplear los caracteres gráficos para construir líneas verticales.

Otra aplicación típica de los caracteres gráficos es la construcción de letras o números de tamaño superior al normal para titulares o cabeceras, como ya se ha visto.



14.3 GRAFICOS DE ALTA RESOLUCION

La pantalla gráfica es un mosaico con varios miles de elementos de imagen

Como hemos visto en los ejemplos anteriores, las posibilidades para realizar dibujos a base de caracteres gráficos son bastante limitadas. Para conseguir dibujos de mayor calidad hay que emplear elementos que sean de mucho menor tamaño que las subdivisiones de un carácter gráfico.

Para entender cómo funcionan los gráficos de alta resolución debemos imaginarnos la pantalla como un mosaico formado por miles de elementos (Ver figura 4). Cada uno de estos elementos tiene un tamaño aproximado de un punto ortográfico (.). En lenguaje técnico, estos puntos se denominan «pixels» o elementos de imagen. Cada uno de los pixels está asociado a un bit de la memoria y puede estar encendido o apagado (blanco o negro) según el valor del bit sea uno o cero. A base de encender o apagar los pixels adecuados construiremos los gráficos y dibujos.

Pixel: Un pixel es el elemento mínimo que se puede dibujar. No es posible dibujar una fracción de pixel.

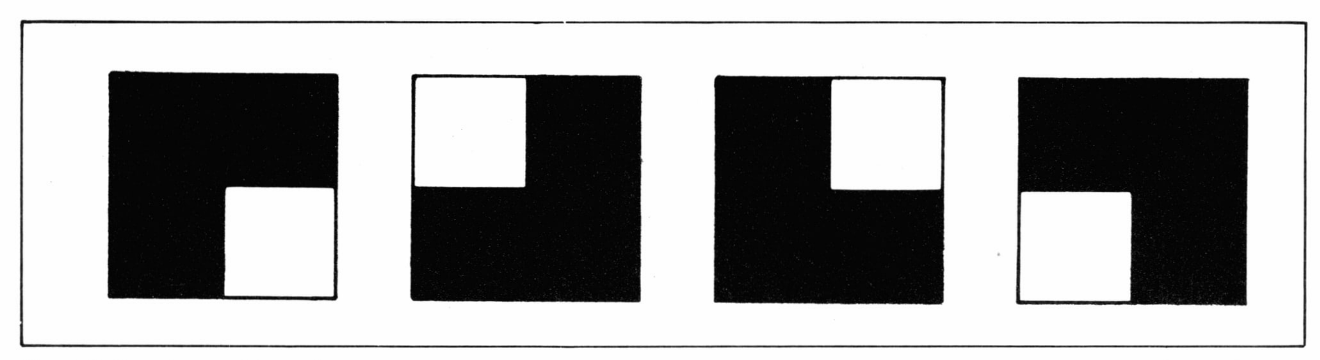


Figura 2 Caracteres gráficos utilizados para dibujar vértices.

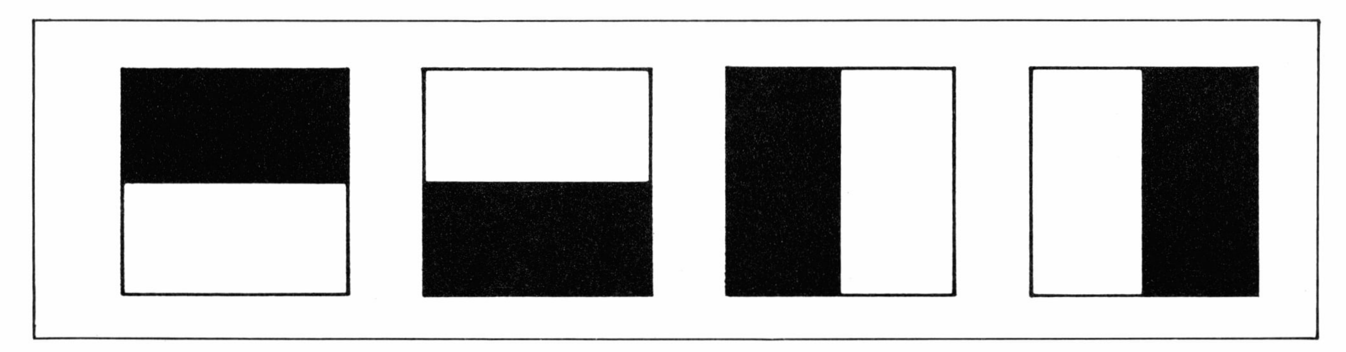


Figura 3 Caracteres gráficos utilizados para trazar líneas horizontales y verticales.

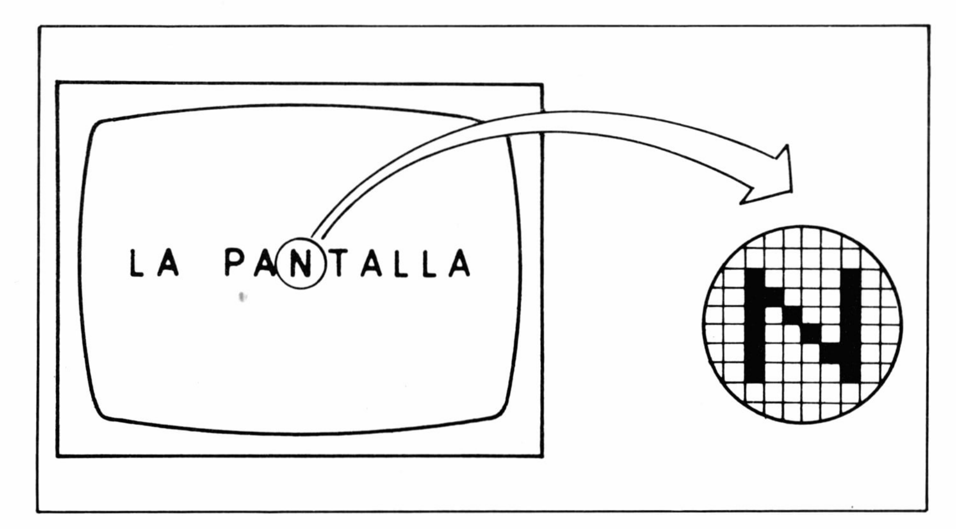


Figura 4 Ampliación de la pantalla, mostrando la estructura de los pixels.

Obviamente, cuanto más pequeños sean los pixels, mayor será la calidad de los gráficos. La calidad de los gráficos se mide por un concepto denominado *resolución*. La resolución es una medida que nos indica hasta qué punto se pueden diferenciar dos posiciones adyacentes. Así, si una pantalla tiene 100 pixels horizontales, cada pixel se diferencia del anterior en un 1 por ciento. Si una pantalla tiene 1000 pixels, entonces cada uno de ellos se diferencia de los contiguos en un 0.1 por ciento. Por esta razón, el grado de resolución de una pantalla se expresa por el producto del número de pixels horizontales por el número de pixels verticales.

Las pantallas más sencillas tienen una resolución de aproximadamente 100×100 puntos. Esto significa que el mosaico tiene unos 100×100 = 10.000 pixels. Aunque este número pueda parecer muy grande, en realidad sólo se pueden conseguir gráficos de una calidad discreta. Las pantallas especializadas en realización de gráficos usadas en ingeniería, topografía, etc. llegan a tener una resolución de 4096×4096 (más de 16 millones de pixels). Esta resolución permite realizar gráficos de calidad fotográfica.

Resolución: La resolución es una medida de la calidad de una pantalla gráfica.

14.3.1 Ejes de coordenadas

A lo largo de este capítulo de gráficos por ordenador iremos repasando algunos conceptos matemáticos que nos serán de utilidad. El primero de estos conceptos son los ejes de coordenadas.

Los ejes de coordenadas son dos rectas que se cortan perpendicularmente (Fig. 5). El punto común a ambas, es decir donde se cortan, se denomina *origen de coordenadas*. El eje horizontal recibe el nombre de *eje de abscisas* y el eje vertical se denomina *eje de ordenadas.* Cada uno de los ejes está graduado. El valor de la graduación empieza en cero en el origen de coordenadas. En el eje de abscisas, la graduación es creciente a la derecha del origen (números positivos) y decreciente a la izquierda (números negativos). En el eje de ordenadas, la graduación es creciente hacia arriba y decreciente hacia abajo. El eje de abscisas se suele denominar también eje X y el de ordenadas eje Y. Asimismo, estos ejes de coordenadas reciben también el nombre de ejes cartesianos en honor del matemático frances R. Descartes.

Los ejes de coordenadas definen cuatro cuadrantes

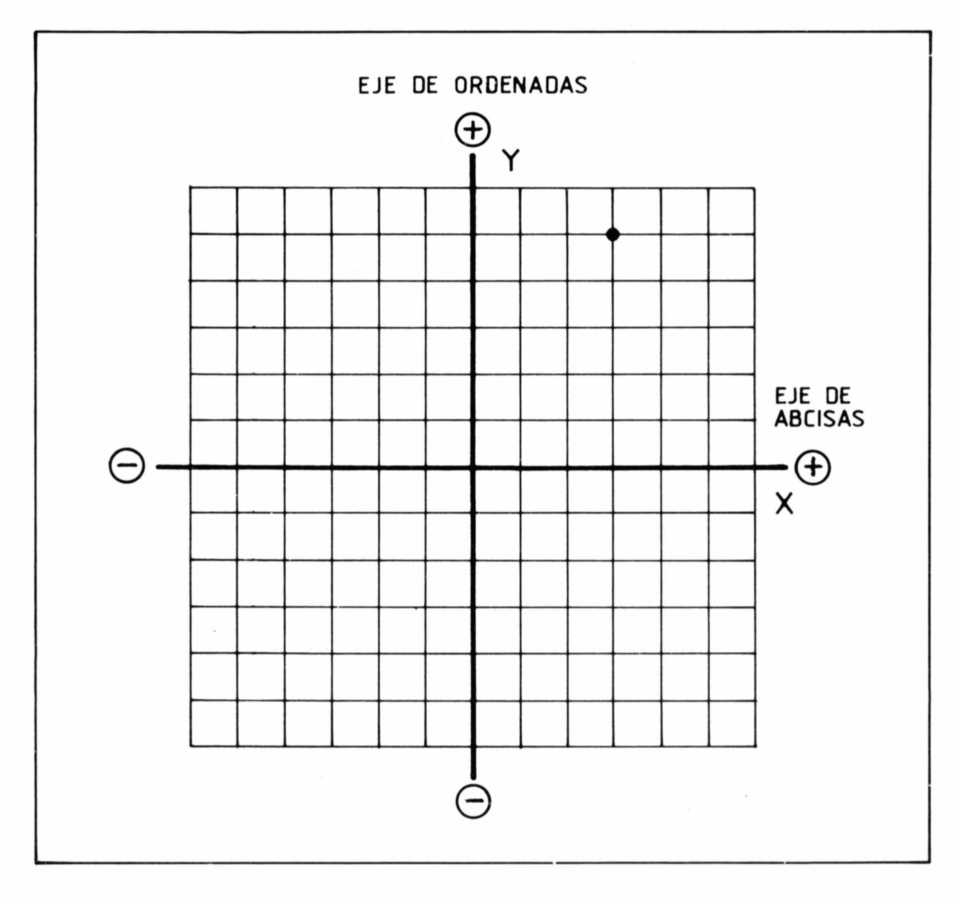
Como se observa en la figura 5, el plano formado por la intersección de las dos rectas queda dividido en cuatro zonas denominadas *cuadrantes*. Las graduaciones de los ejes definen un «enrejado» por toda la superficie del plano. Cualquier punto del plano se puede definir mediante dos valores. El primer valor es su distancia al eje vertical (ordenadas) y se denomina abscisa del punto. Este valor se suele representar por la letra *x* minúscula. El segundo valor es su distancia al eje horizontal (abscisas) y se denomina ordenada del punto. Este valor se suele representar por la letra *y* minúscula. Los dos valores juntos, abscisa y ordenada, reciben el nombre de *coordenadas* de un punto. Las coordenadas de un punto definen con exactitud su posición en el plano. En general se suele indicar primero el valor de las abscisas y luego el de las ordenadas. Por tanto, el punto (3,5) estará situado tres posiciones hacia la derecha y 5 hacia arriba. Es el punto que aparece marcado en la figura 5.

Las coordenadas son únicas para cada punto

De los cuatro cuadrantes, el superior derecho es el único cuyos puntos tienen siempre sus coordenadas positivas.

Como es fácil de adivinar, los ejes de coordenadas están estrechamente ligados con la estructura de mosaico de la pantalla gráfica. En efecto, se puede considerar a la pantalla como *el cuadrante superior derecho* de

Figura 5 *Ejes de coordenadas con indicación de las zonas positivas y negativas de los ejes X e Y.*



La pantalla gráfica equivale al cuadrante superior derecho de unos ejes de coordenadas



unos ejes de coordenadas. Por tanto, el pixel situado en el *vértice inferior izquierdo* de la pantalla tendrá las coordenadas (0,0). La línea horizontal inferior constituirá el eje X y la línea vertical izquierda, el eje Y. La graduación de los ejes está formada por unidades de pixel. Una pantalla de 100×200 tendrá un eje de abscisas con 200 graduaciones y un eje de ordenadas de 100 graduaciones. Hay que recalcar que el elemento mínimo de imagen es el pixel y que no es posible dibujar una fracción de pixel. Por esta razón, no pueden existir graduaciones intermedias. En consecuencia, no pueden existir unas coordenadas con números fraccionarios.

Es conveniente que todo el vocabulario que ha aparecido en este recordatorio matemático (abscisas, coordenadas, cuadrantes, etc.) no sea olvidado, puesto que haremos un uso constante del mismo.

14.3.2 Operaciones primitivas

Una vez conocida la estructura de la pantalla, veamos ahora cómo podemos dibujar en ella. En realidad, la única operación que podemos hacer es cambiar el estado de un pixel. De modo que, para trazar una línea, ac-

tivaremos todos los pixels que se encuentren en el camino entre el origen y el final de la recta.

Todo gráfico o dibujo se puede realizar a base de combinar una serie de operaciones elementales llamadas operaciones *primitivas*. Desafortunadamente, no existen unas instrucciones normalizadas en BASIC (ni de hecho en ningún otro lenguaje de ordenador) para efectuar estas operaciones primitivas. Para obviar este problema y poder elaborar programas reales, nos definiremos nuestras propias instrucciones. Por las «Prácticas con el microordenador» correspondientes a este capítulo, ya sabe cómo se escriben en realidad estas instrucciones en nuestro ordenador. Bastará entonces sustituir la instrucción escrita según nuestra convención por la instrucción o grupos de instrucciones reales.

Las operaciones primitivas son las siguientes:

a) Posicionamiento de un pixel:

Esta operación nos permite activar un pixel en concreto indicando sus coordenadas. Un efecto colateral de esta operación es que las coordenadas de este punto pasan a ser las coordenadas de lo que se denomina *punto actual*. Para nosotros, la instrucción asociada a esta operación primitiva tendrá la forma:

```
PUNTO(X, Y)
```

en donde X e Y serán los valores de la abscisa y ordenada del punto. Según esta convención, la instrucción

```
PUNTO(3, 5)
```

activará el pixel de coordenadas (3,5).

b) Trazado de una línea:

Esta operación activa todos los pixels que se encuentren en línea recta desde el punto actual hasta el punto cuyas coordenadas le indiquemos. Este último punto pasa a ser ahora el punto actual.
En nuestro lenguaje convencional, esta instrucción tendrá la forma:

```
LINEA(X, Y)
```

en donde X e Y son las coordenadas del punto final de la recta.

c) Posicionamiento de un carácter:

Esta operación sitúa un carácter en las coordenadas que se indiquen. Se utiliza para colocar rótulos sobre un gráfico. En general, sólo disponen de

esta operación los ordenadores especializados en gráficos. Por esta razón no la utilizaremos en nuestros programas. En su lugar emplearemos la función AT que, aun teniendo menos posibilidades, será suficiente para nuestros fines.

Las operaciones primitivas se basan en la activación o desactivación de un pixel

Operaciones primitivas: Son aquellas operaciones gráficas elementales. A base de combinar varias de ellas se puede construir cualquier gráfico.

En todos los programas que aparezcan en esta unidad supondremos que la pantalla tiene unas dimensiones de 100 para el eje vertical (ordenadas) y de 200 para el eje horizontal (abscisas). En caso necesario, en el capítulo de «Prácticas con el microordenador» se indicarán las modificaciones que habrá que realizar para adaptar el programa a las dimensiones de la pantalla de nuestro ordenador.

14.3.3 Pendiente de las líneas rectas

La pendiente de una línea recta es su inclinación respecto al eje de abscisas.

Como ya sabemos, una línea recta se construye a base de activar los pixels que se encuentran sobre la trayectoria de dicha recta. Si las líneas son horizontales o verticales, el dibujo sale perfecto, puesto que coincide con la disposición de los pixels sobre el mosaico. Si la recta forma un ángulo de 45 grados, la línea dibujada también tiene bastante calidad pues se activan los pixels situados en diagonal. En el caso de rectas con pendientes casi horizontales o casi verticales, el dibujo tiene mucha menos calidad y se observan discontinuidades.

El siguiente programa ilustra este efecto.

Programa de «Pendientes rectas»

```
10 REM Pendientes rectas
20 FOR A=0 TO 1.58 STEP 0.157
30   PUNTO(0,0)
40   LET X=100*COS(A)
50   LET Y=100*SIN(A)
60   LINEA(X,Y)
70   NEXT A
```

De momento, hasta que no veamos el apartado dedicado a Trigonometría, podemos pasar por alto los cálculos que realiza el programa, en donde se emplean las funciones COS y SIN. Basta con saber que dibuja once rectas con pendientes que varían entre la horizontal y la vertical.

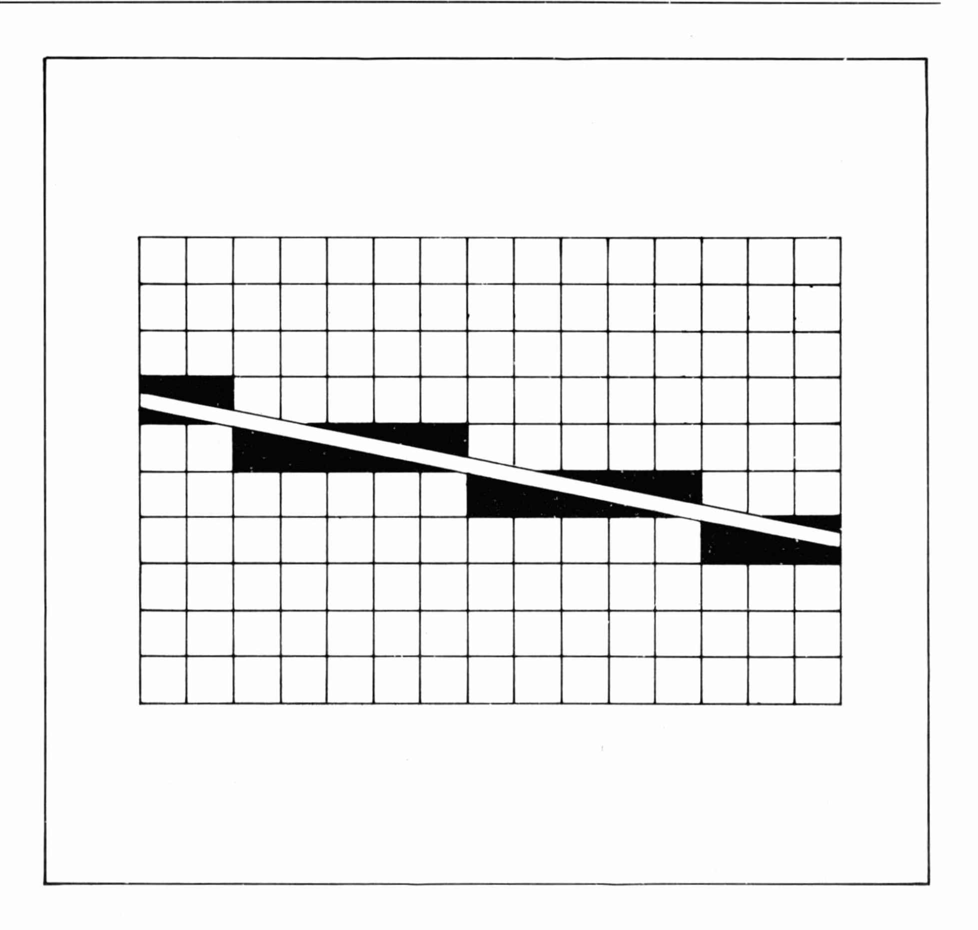


Figura 6 Forma de trazar en un ordenador una recta. La línea blanca es la recta teórica. Los recuadros negros son los pixels activados que intentan reproducir la recta.

En la pantalla se observa el efecto mencionado. Aparecen discontinuidades en las rectas. Esto es lógico si pensamos que no se puede activar una fracción de pixel. O se activa todo o no se activa nada. Entonces, cuando la línea recta cruza la frontera de dos pixels, la activación pasa al pixel adyacente provocando la discontinuidad. En la figura 6 hay un esquema de este efecto.

Este problema es común a todos los sistemas gráficos basados en ordenadores. En los sistemas de alta resolución, al tener los pixels un tamaño muy reducido, este efecto, aunque sigue estando presente, no es apreciable por el ojo humano.

14.3.4 Operaciones auxiliares

Las operaciones auxiliares, aunque no son imprescindibles, facilitan la realización de gráficos complejos

Aparte de las operaciones primitivas que ya hemos descrito, muchos ordenadores incorporan también algunas operaciones auxiliares. Estas operaciones auxiliares se utilizan para dibujar polígonos, círculos, elipses, etc. Sin embargo, ya hemos comentado que, a partir de operaciones primitivas, se puede construir cualquier gráfico. Por tanto, todas las operaciones auxiliares las podemos realizar también a base de combinar una serie de operaciones primitivas. No obstante, para saber cómo construir esta serie de operaciones, se precisan unos conocimientos de Trigonometría.

En el siguiente apartado efectuaremos un repaso de las nociones fundamentales de Trigonometría. Si ya se conocen bien, puede omitirse el estudio de este apartado y continuar con los gráficos bidimensionales. Si es la primera vez que estudiamos Trigonometría, es posible que nos parezca un tema algo difícil. En este caso es aconsejable realizar una lectura superficial y seguir con los demás apartados. Entonces, cada vez que aparezca un concepto de Trigonometría desconocido, retrocederemos a este apartado y lo estudiaremos con detenimiento. De hecho, los programas funcionarán igual aunque no se entienda el fundamento teórico en que se basan. Evidentemente, el problema estará si usted quiere realizar un programa en el que tenga que manejar estos conceptos, en cuyo caso conviene que los profundice un poco más.

14.3.5 Trigonometría

La Trigonometría es una parte de las Matemáticas que estudia las relaciones entre los segmentos (lados) de un triángulo y los ángulos correspondientes.

Daremos ahora unas nociones elementales, puesto que el objetivo de esta Enciclopedia no son las Matemáticas. Sin embargo, queremos advertirle que el no asimilarlas completamente no le impedirá seguir esta obra aunque le servirán para comprender mejor algunas instrucciones, que verá principalmente en los programas de gráficos. Por eso, si no las conoce, le recomendamos que intente asimilarlas en la medida de lo posible.

Ya vimos lo que eran unos ejes de coordenadas y en la figura 5 vio usted cómo se dibujan esos ejes. Ahora vamos a dibujar una circunferencia sobre esos ejes, tomando como centro de la circunferencia el punto donde se cruzan los ejes. Verá que los ejes forman cuatro radios perpendiculares entre sí, según se muestra en la figura 7.

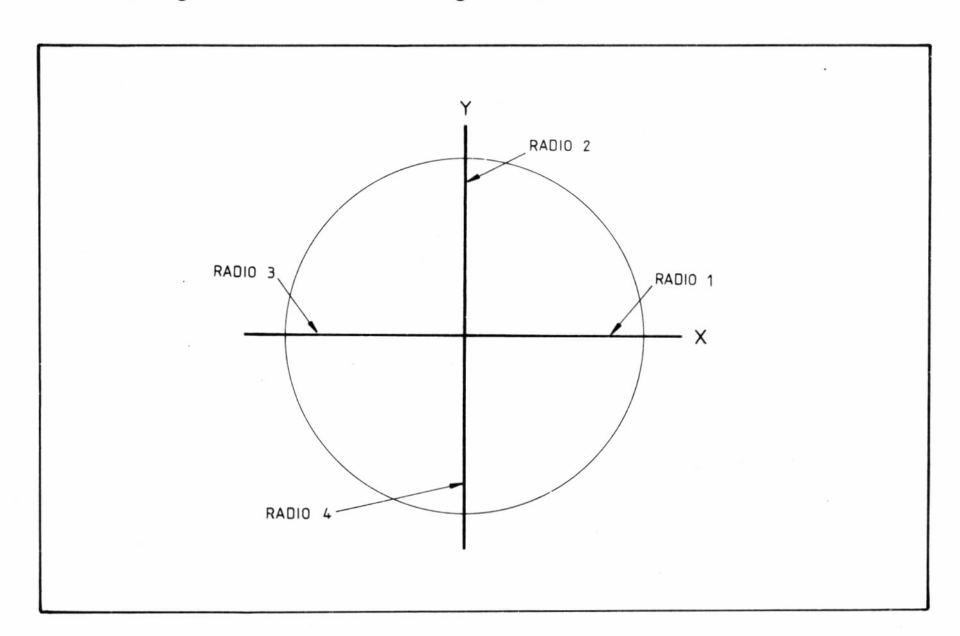
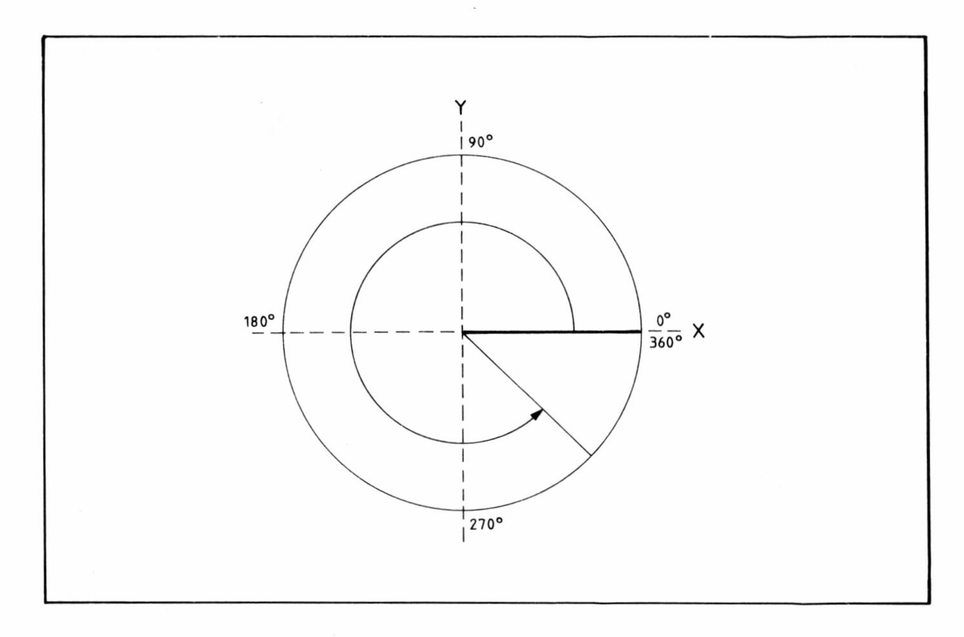


Figura 7 Trazado de una circunferencia sobre un eje de coordenadas.

Figura 8 Movimiento de un radio a lo largo de la circunferencia.



Ahora vamos a suponer que existe un quinto radio, que estaría en este momento superpuesto sobre el eje X, en el lugar que ocupa el radio 1. Este radio se podría mover a los largo de la circunferencia como lo harían las agujas de un reloj, pero en sentido contrario. En la figura 8 se ha representado el movimiento de este quinto radio. En esta figura hemos dibujado los ejes con trazo discontinuo para que vea mejor la trayectoria y el punto de partida de este quinto radio.

El desplazamiento lo mediremos mediante el ángulo que forma este quinto radio respecto al eje X de donde parte. Por tanto, cuando el radio está superpuesto al eje X, el ángulo vale cero. Cuando está perpendicular, valdrá 90 grados; cuando llega de nuevo al eje X valdrá 180 grados; cuando llegue de nuevo al eje Y valdrá 270 y al llegar al origen de donde partió valdrá 360 grados. Es decir, los grados de la circunferencia.

El radián es una unidad de medida de los ángulos

Normalmente, en Trigonometría los ángulos no se miden en grados sino en *radianes*. Por eso veremos lo que es un radián y su equivalencia en grados.

Comenzaremos definiendo lo que es un radián.

Un radián se define como el ángulo cuyo arco de circunferencia es igual a la longitud del radio. En la figura 9 tiene dibujado un ángulo sobre una circunferencia cuyo valor es de un radián. Es decir, cuyo arco tiene el valor del radio.

Recuerde que la longitud de la circunferencia es igual al radio multiplicado por 2π; es decir, 2 multiplicando por 3,1416 aproximadamente, que es el valor de PI (π). Por tanto, el valor de la circunferencia sería:

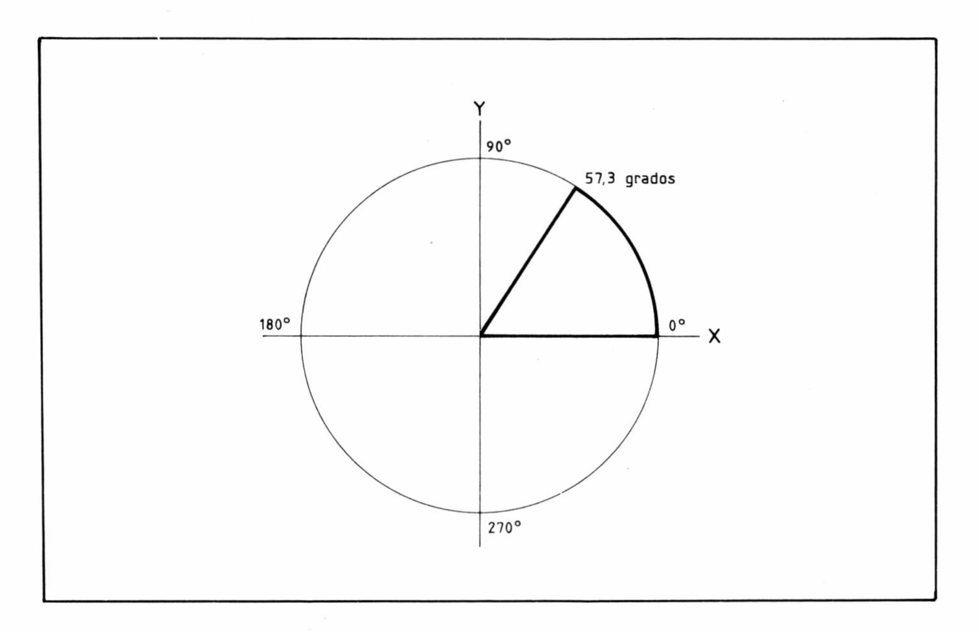
$$R \times 2\pi$$

donde *R* es el valor del radio.

Suponiendo que el radio vale 1, esto sería igual a:

$$1 \times 2 \times 3{,}1416$$

Figura 9 Angulo cuyo valor es un radián.



Esto nos daría que la circunferencia medida en radianes sería igual a:

$$2\pi$$

Con esto ya podemos calcular fácilmente el valor de un grado medido en radianes o el valor de un radián medido en grados.

Un grado es igual a:

$$\frac{2\pi}{360} = \frac{2 \times 3{,}1416}{360} = 0{,}01745 \text{ radianes}$$

o lo que es lo mismo:

$$\frac{\pi}{180} = \frac{3{,}1416}{180} = 0{,}0745 \text{ radianes}$$

Un radián es igual a:

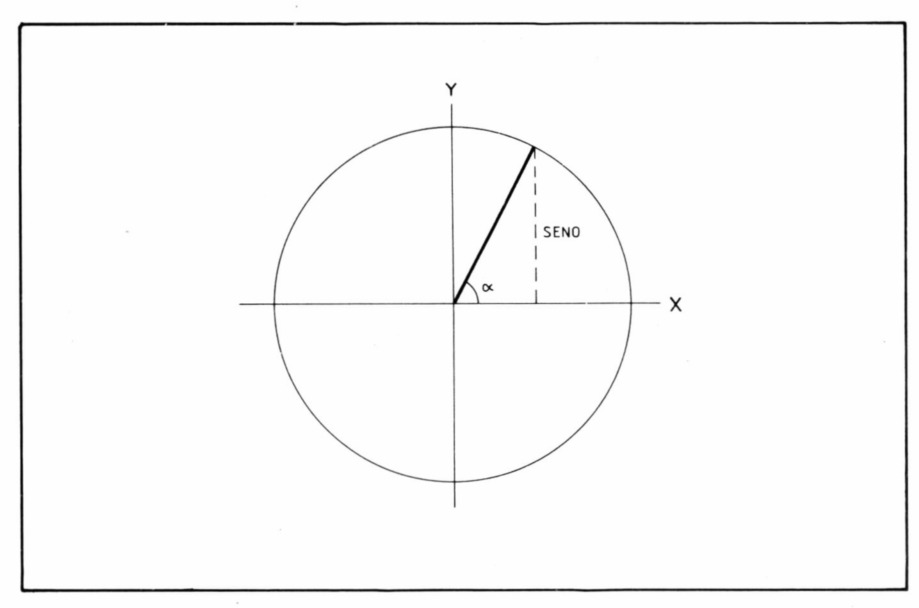
$$\frac{360}{2\pi} = \frac{360}{2 \times 3{,}1416} = 57{,}3 \text{ grados}$$

o lo que es lo mismo:

$$\frac{180}{\pi} = \frac{180}{3{,}1416} = 57{,}3 \text{ grados}$$

Supongamos ahora que, desde el punto hasta el que se ha desplazado el radio en la figura 9, trazamos una recta perpendicular al eje X, según se muestra en la figura 10. (La recta está en trazo discontinuo para que se vea más claramente.) Con ello hemos formado un triángulo rectángulo cuyas

Figura 10 El seno de un ángulo.



propiedades vamos a ver enseguida y que constituyen el núcleo fundamental de la Trigonometría.

El valor del seno oscila entre 1 y –1

Tenemos en primer lugar el lado vertical (lado de línea discontinua) del triángulo que va desde el punto donde el radio corta a la circunferencia hasta el eje X. La longitud de este segmento o lado del triángulo se llama *seno* del ángulo *a*.

Observe que si hace bajar el radio que está dibujado con trazo más fuerte hacia el eje X, la línea de trazo discontinuo, es decir el seno, será cada vez más corta. Así, si lo va desplazando hasta dejarlo sobre el eje X, el seno valdrá cero. Por eso, a medida que el ángulo crece, también aumenta el valor del seno hasta que llega a los 90 grados y se coloca sobre el eje Y, en cuyo caso vale uno, es decir, el valor de un radio, pues, es igual a él. Si sigue haciendo girar el radio hacia la izquierda comenzará a disminuir hasta llegar de nuevo al eje X (180 grados), en que de nuevo vale cero. Para ángulos superiores a 180 grados el seno es negativo. Así cuando llega a los 270 grados, el seno vale menos uno (–1). Finalmente al llegar a los 360 grados el seno vale otra vez cero.

Ya vimos en el capítulo dedicado a las funciones que el seno de un ángulo se escribe:

SIN(a)

La palabra SIN proviene del latín «sinus» y en inglés se ha mantenido el mismo nombre para significar el seno.

Recordemos ahora que el valor de un ángulo debe estar expresado en radianes. Los siguientes ejemplos muestran los valores que da la función seno (SIN). El valor de π se ha representado como PI.

```
PRINT SIN(0)          escribe 0
PRINT SIN(1)          escribe 0.84147088
PRINT SIN(PI/2)       escribe 1
PRINT SIN(3*PI/2)     escribe -1
```

Aunque el cálculo de las funciones trigonométricas es un poco complicado y por eso se acude a las tablas que ya están calculadas o a las calculadoras que dan directamente el resultado, para que comprenda un poco el significado de los valores que acabamos de citar le diremos lo que significan los dos primeros.

En el primer caso,

```
PRINT SIN(0)        escribe cero
```

quiere decir que cuando el ángulo del triángulo vale cero (SIN 0) el seno vale también cero. Es el caso que poníamos más arriba de cuando el quinto radio se encuentra sobre el eje X.

En el segundo caso,

```
PRINT SIN(1)        escribe 0.84147088
```

quiere decir que cuando el ángulo del triángulo vale un radián (es igual al valor del radio), el seno (es decir, la línea discontinua que veíamos en la figura 10) es igual a 0,84147088, el valor del radio. Es decir, vale un poco menos que el radio, como puede verlo en la misma figura 10.

Por estos ejemplos ya puede deducir usted el significado de los demás.

Tomemos ahora de nuevo el mismo triángulo de la figura 10 y que reproduciremos en la figura 11. El segmento del triángulo que está superpuesto al eje X se denomina *coseno* de a. En la figura 11 puede ver el valor del coseno en línea discontinua. El coseno tiene un comportamiento inverso al del seno. En efecto, cuando el ángulo es cero, puesto que el segmento

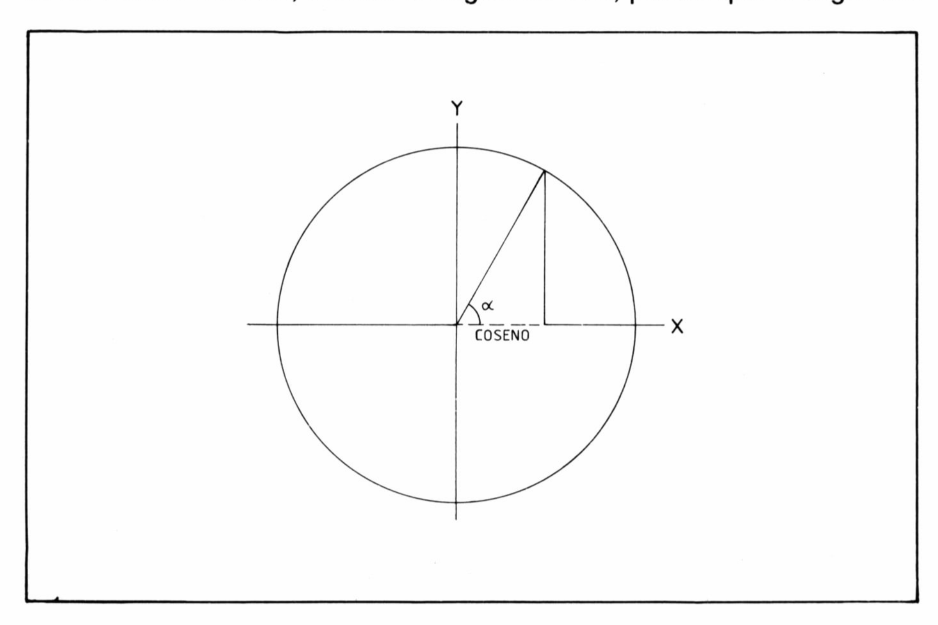


Figura 11 El coseno de un ángulo.

coincide con el radio, el valor del coseno es 1. A medida que el ángulo crece, el valor del coseno disminuye hasta que para un ángulo de 90 grados el coseno vale cero.

Los argumentos de las funciones trigonométricas estarán siempre en radianes

A partir de este momento, el valor del coseno es negativo, ya que el segmento está a la izquierda del eje X. Cuando el ángulo vale 180 grados, el coseno vale menos uno (–1). Finalmente, al llegar a 360 grados (2π radianes), el coseno vale uno de nuevo.

En BASIC la función coseno de *a* se escribe:

COS(a)

El valor de *a* debe estar, como siempre, expresado en radianes. Veamos algunos ejemplos:

```
PRINT COS(0)         escribe 1
PRINT COS(1)         escribe 0.540302
PRINT COS(PI/2)      escribe 0
PRINT COS(PI)        escribe -1
```

Al igual que al estudiar el caso del seno, lo que se escribe entre paréntesis después de la función COS es el valor del ángulo medido en radianes. Y el resultado que se obtiene es el valor del coseno.

Hay todavía una función importante en Trigonometría. Vea la figura 12. El radio se ha prolongado hasta interceptar la recta perpendicular al eje y que toca la circunferencia por su parte exterior. Así se ha construido un nuevo triángulo formado por el radio prolongado, la línea discontinua en la figura y el eje X. La longitud de esta línea discontinua es lo que se llama, precisamente, *tangente de a.*

El valor de la tangente es cero cuando el ángulo vale cero. A medida que el ángulo crece, el valor de la tangente aumenta con rapidez. Para

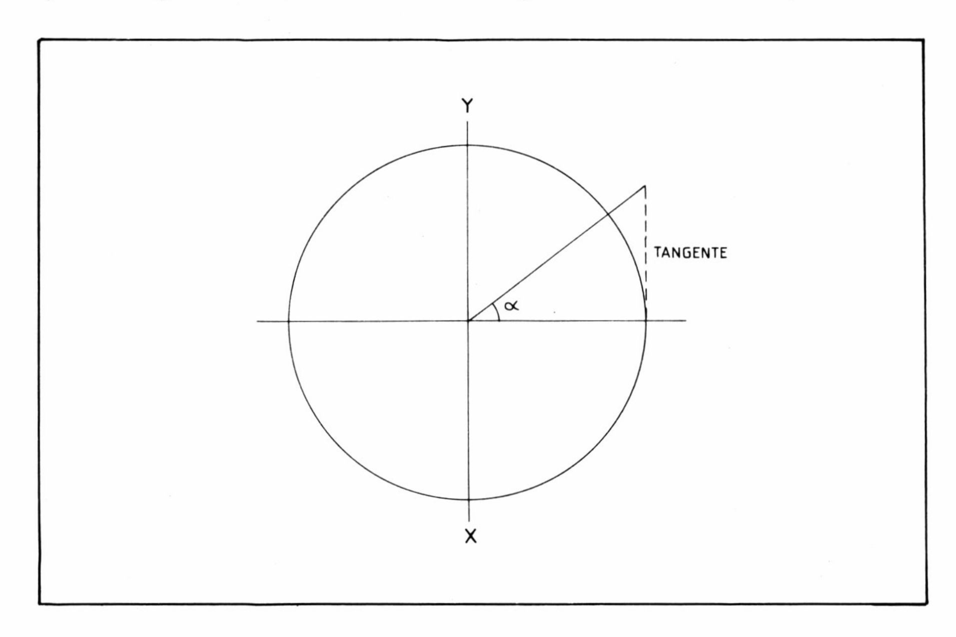


Figura 12 La tangente de un ángulo.

ángulos próximos a los 90 grados la tangente vale casi infinito, pues es prácticamente una paralela. La tangente también se puede definir como la división del seno por el coseno de un ángulo.

$$\text{tangente} = \frac{\text{seno}}{\text{coseno}}$$

En BASIC, la función tangente de *a* se escribe:

TAN(a)

donde el valor de *a* viene dado en radianes. Veamos algunos ejemplos:

```
PRINT TAN(0)        escribe 0
PRINT TAN(1)        escribe 1.5574
PRINT TAN(2*PI)     escribe 0
```

Para algunos ángulos muy pequeños, el valor de la tangente es aproximadamente igual al del ángulo (medido en radianes). Por ejemplo:

```
PRINT TAN(0.001) escribe 0.001
```

Esto es fácil de comprender si pensamos en la definición. Cuando el ángulo es muy pequeño, el arco de circunferencia tiene aproximadamente la misma longitud que el segmento de tangente. Puesto que el radián mide precisamente la longitud del arco de circunferencia, es lógico que el valor del arco sea igual al de la tangente.

Existen además otras funciones que son la cotangente, secante y cosecante, pero que carecen de interés para nuestros fines. Además su valor se puede obtener a partir de las tres anteriores. En cambio, sí que tienen interés las funciones inversas del seno, coseno y tangente.

Las funciones inversas se utilizan cuando conocemos el valor de la función directa y queremos encontrar el valor del ángulo correspondiente. Las tres funciones inversas son arco seno, arco coseno y arco tangente.

En el BASIC estándar, sólo está definida como función el arco tangente. Se escribe de la forma:

```
ATN(X)
```

y el resultado es el ángulo (o arco) cuya tangente es X.

Naturalmente el resultado viene expresado en radianes. Veamos algunos ejemplos:

```
PRINT ATN(0)        escribe 0
PRINT ATN(1E+10)    escribe 1.57079 (π/2)
```

Como acabamos de decir, las funciones TAN y ATN son la inversa una de otra. Por tanto,

```
PRINT ATN(TAN(1))    escribe 1
PRINT TAN(ATN(1))    escribe 1
```

La función ATN efectúa la operación inversa de TAN

Hay que tener en cuenta que este cálculo sólo es cierto si el ángulo pertenece al primer cuadrante. En caso contrario, se obtiene el ángulo complementario.

Algunos ordenadores incorporan también el arco seno y el arco coseno. Si no es así, podemos obtenerlos a partir del arco tangente según las siguientes fórmulas

```
arc sen(X) = ATN(X/SQR(1-X*X))
arc cos(X) = ATN(SQR(1-X*X)/X)
```

Puesto que la raíz cuadrada da siempre un valor positivo, estas fórmulas sólo son válidas para el primer cuadrante.

RESUMEN

Los caracteres gráficos cuyos códigos son superiores a 127 se utilizan para construir tablas y dibujos de tipo lineal. Están divididos en varias partes, cada una de las cuales va asociada a un bit del octeto de la memoria donde se almacena el carácter.

Estos caracteres se pueden obtener mediante la función CHR$ y, en algunos ordenadores, directamente del teclado.

La pantalla gráfica de alta resolución se puede considerar como un mosaico que contiene miles de elementos de imagen denominados «pixels». La resolución es una medida de calidad de una pantalla gráfica. Para trazar gráficos se considera que la pantalla es equivalente al cuadrante superior derecho de unos ejes de coordenadas. En consecuencia, cada pixel se expresará mediante dos valores denominados coordenadas. A causa de la estructura de los pixels, estas coordenadas nunca pueden ser fraccionarias.

Las operaciones primitivas nos permiten construir cualquier gráfico. Estas operaciones son: colocar un punto, trazar una recta y dibujar un carácter. Estas operaciones memorizan siempre las coordenadas del punto actual. Este punto es el último pixel sobre el que se ha actuado. Muchos ordenadores incorporan operaciones auxiliares que están construidas combinando varias operaciones primitivas. Debido a la limitación que impone el tamaño de un pixel y a que no se puede activar una fracción de pixel, algunas rectas tendrán discontinuidades en su trazado.

En BASIC existen las funciones trigonométricas SIN que equivale al seno, COS que equivale al coseno y TAN que equivale a la tangente. El valor del ángulo debe estar expresado siempre en radianes. En el BASIC estándar, la única función inversa definida es ATN que es el arco tangente. Su resultado está dado en radianes.

EJERCICIOS DE AUTOCOMPROBACION

Complete las frases siguientes:

1. Los caracteres gráficos tienen un número de código ASCII superior a ....

2. Mediante la función .... podremos obtener cualquier símbolo definido en el ordenador.

3. Un elemento de imagen recibe también el nombre técnico de ....

4. La .... nos indica el grado de calidad de una pantalla gráfica.

5. El estado de un .... de la memoria indicará si un pixel está activado o desactivado.

6. El eje de .... es el eje horizontal de unos ejes de coordenadas.

7. Cualquier punto situado en unos ejes de coordenadas se puede definir mediante .... valores.

8. El pixel situado en el vértice .... de la pantalla tendrá las coordenadas (0,0).

9. La graduación de los ejes de la pantalla está formada por .... de pixel.

10. Mediante las operaciones .... se puede construir cualquier gráfico.

11. El último punto sobre el que se ha actuado recibe el nombre de .... y sus coordenadas quedan memorizadas.

12. La operación de trazado de una línea funciona a base de activar los .... que se encuentran entre los dos puntos especificados.

13. Según sea la .... de una recta, pueden aparecer discontinuidades en su trazado en la pantalla.

14. Las discontinuidades son menos perceptibles en pantallas que tengan .... resolución.

15. Las operaciones auxiliares se pueden obtener a base de combinar una serie de operaciones ....

16. En BASIC, el argumento de las funciones SIN, COS y TAN viene expresado en ....

17. El resultado de las funciones SIN y COS está siempre comprendido entre 1 y ....

18. El valor de la función TAN .... muy rápidamente a medida que aumenta el valor del ángulo.

19. Para ángulos muy pequeños (medidos en radianes) el valor de la tangente vale aproximadamente lo mismo que ....

20. La función ATN es la función .... de TAN.

Encierre en un círculo la respuesta que corresponda a la alternativa correcta.

21. Los recuadros de pantalla ocupados por un carácter gráfico o por un carácter normal son:

a) Del mismo tamaño.

b) Los gráficos son el doble de anchos.

c) Los gráficos son el doble de altos.

d) Depende de cada carácter.

22. Entre dos caracteres adyacentes en la pantalla:

    a) Existe un pixel de separación.
    b) Existe un recuadro entre ellos.
    c) No hay separación alguna.
    d) La separación depende de los caracteres.

23. Un carácter gráfico se divide en

    a) 16 partes.
    b) 4 partes.
    c) No se divide.
    d) Depende de cada modelo de ordenador.

24. La resolución de una pantalla gráfica se suele medir en

    a) Pulgadas.
    b) Centímetros cuadrados.
    c) Número de pixels horizontales por número de pixels verticales.
    d) Número total de caracteres que puede contener.

25. En el eje de abscisas, la graduación es

    a) Decreciente hacia arriba.
    b) Creciente hacia la izquierda.
    c) Creciente hacia la derecha.
    d) Decreciente hacia abajo.

26. El único cuadrante que tiene todas las coordenadas positivas es el

    a) Superior izquierdo.
    b) Superior derecho.
    c) Inferior izquierdo.
    d) Inferior derecho.

27. La máxima precisión con que podemos especificar unas coordenadas es de

    a) Un pixel.
    b) Un carácter.
    c) Medio pixel.
    d) Un milímetro.

28. Las operaciones primitivas se basan en

    a) Trazar rectas.
    b) Trazar curvas.
    c) Activar o desactivar los pixels.
    d) Escribir un carácter.

29. Las discontinuidades en las líneas rectas son originadas:

    a) Porque los pixels están separados.
    b) Porque los pixels se activan por grupos.
    c) Porque las rectas tienen pendientes erróneas.
    d) Porque no se pueden dibujar fracciones de pixel.

30. Para convertir un ángulo en grados a un ángulo en radianes, lo multiplicamos por

    a) $\pi/180$
    b) $\pi/360$
    c) $180/\pi$
    d) 2π



14.4 GRAFICOS BIDIMENSIONALES

En la pantalla de un ordenador se pueden representar dibujos en dos y tres dimensiones. Los gráficos bidimensionales se pueden representar directamente puesto que la pantalla también tiene dos dimensiones. Por el contrario, los gráficos tridimensionales no se pueden representar directamente. En su lugar hay que representar proyecciones o perspectivas de los mismos. Por esta razón estudiaremos en primer lugar los gráficos bidimensionales.

14.4.1 Trazado de polígonos y circunferencias

Los polígonos regulares son aquellos que tienen todos los lados y ángulos iguales. Para construirlos nos aprovecharemos del hecho de que para los polígonos regulares siempre existe una circunferencia donde pueden ser inscritos. Según esto, para trazar un polígono de *n* lados, bastará con dividir la circunferencia en *n* trozos iguales. Realizar esta división puede parecernos un problema algo difícil. Pero si aplicamos los conocimientos de Trigonometría, entonces la solución es prácticamente inmediata.

Tomemos como ejemplo un hexágono (6 lados). Los vértices de este polígono han de estar a la misma distancia unos de otros a lo largo de la circunferencia. Si la circunferencia tiene un total de 360 grados, entre vértice y vértice habrá 360/6=60 grados. Si suponemos que un vértice está situado sobre el eje X (ángulo igual a cero), el siguiente estará a 60 grados (ver figura 13), el siguiente a 120, etc. Las coordenadas del vértice serán precisamente el seno y el coseno de 60 grados según las definiciones trigonométricas que vimos antes.

El dibujo de los polígonos regulares es fácil de programar, pues se pueden inscribir en un círculo

El razonamiento anterior es aplicable a todos los polígonos regulares y lo único que varía es el ángulo entre vértices. El siguiente programa lo utilizaremos para trazar *cualquier polígono regular.*

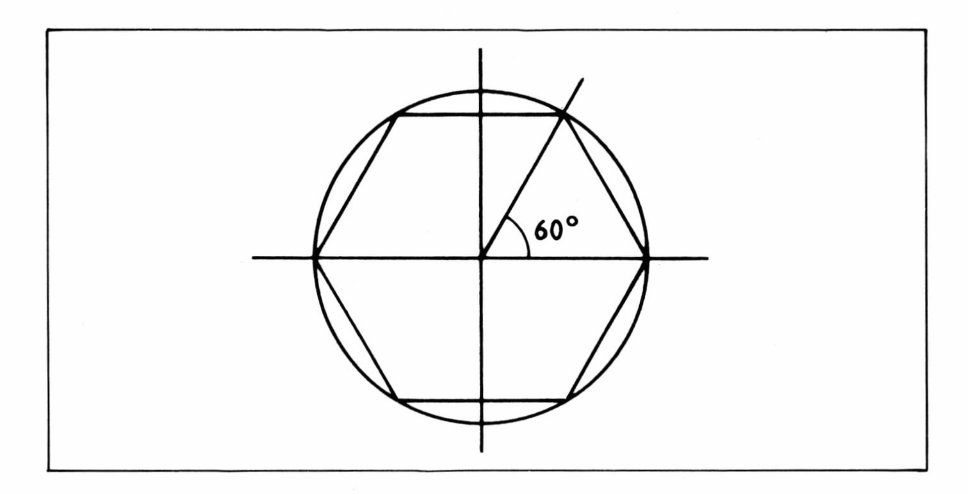
Programa de «Trazado de polígonos»

```
 10 REM Trazado poligonos
 20 INPUT "NUMERO DE LADOS:",N
 30 LET F=2*PI+1E-4
 40 LET D=2*PI/N
 50 LET XO=100: LET YO=50: LET R=45
 60 PUNTO(XO+R,YO)
 70 FOR A=0 TO F STEP D
 80   LET X=XO+R*COS(A)
 90   LET Y=YO+R*SIN(A)
100   LINEA(X,Y)
110   NEXT A
120 STOP
```

En primer lugar, el programa (línea 20) pregunta el número de lados que tendrá el polígono. La variable F de la línea 30 contiene la longitud total de la circunferencia. Se supone que PI es el equivalente de π. Se le añade un número muy pequeño (1E-4) para evitar problemas de redondeo de decimales cuando el número de lados N no es un divisor exacto de 360. En la línea 40 se calcula el desplazamiento (en radianes) que hay entre vértice y vértice.

Todos los razonamientos geométricos hechos hasta este momento partían de la base de que la circunferencia estaba centrada en el punto (0,0). Esta suposición no es viable en la práctica puesto que, como sabemos, en la pantalla sólo se puede representar el primer cuadrante. Sin em-

Figura 13 Construcción de un exágono a base de dividir la circunferencia en seis arcos de 60 grados.



bargo, esto no constituye un inconveniente grave. Basta con desplazar el centro de la circunferencia hasta el centro de la pantalla. Esta operación es la que se realiza en la línea 50, donde se establecen las coordenadas del punto central (X0,Y0). Además, se establece también el valor del radio de la circunferencia.

En la línea 60 colocamos el primer vértice (para ángulo igual a 0). La línea 70 establece un bucle donde el ángulo A varía desde cero hasta el límite contenido en F, con incrementos de D radianes. En las líneas 80 y 90 se calculan las coordenadas de los vértices. Puesto que el radio no es la unidad, hay que multipicar el seno y el coseno por el radio, además de sumarle el desplazamiento del centro. La línea 100 traza el lado entre vértices y finalmente la línea 110 cierra el bucle.

Un polígono regular con muchos lados equivale a la circunferencia

Podemos emplear este programa para construir triángulos, cuadrados, pentágonos, etc. Observaremos que, cuando el número de lados es grande (unos 30 o 40), el polígono coincide prácticamente con un círculo. Este hecho no debe sorprendernos si pensamos que un círculo se puede considerar como un polígono de infinitos lados. Dada la resolución de la pantalla, no hace falta utilizar demasiados lados. Normalmente con unos 50 se obtienen círculos aceptables. Esto nos hace pensar que podemos modificar el programa anterior para obtener círculos directamente, en caso de que nuestro ordenador no incorpore la operación auxiliar de trazado de círculos.

El programa queda de la siguiente forma:

Programa de «Trazado de círculos»

```
10 REM Circulo
20 LET XO=100: LET YO=50: LET R=45
30 PUNTO(XO+R, YO)
40 FOR A=0 TO 6.284 STEP 0.1571
50   LET X=XO+R*COS(A)
60   LET Y=YO+R*SIN(A)
70   LINEA(X,Y)
80   NEXT A
90 STOP
```

En la línea 20 se establece el radio y el centro del círculo que pueden ser variados a voluntad. El resto del programa no sufre variación. Notemos que 6.284 es el valor aproximado de 2π.



14.4.2 Elipses y espirales

El programa utilizado para trazar círculos se puede adaptar fácilmente para dibujar elipses y espirales.

Si los dos semirradios son iguales se traza una circunferencia

En una circunferencia, el radio es constante. En cambio, en una elipse podemos considerar que existen dos radios perpendiculares de longitud distinta, según se ve en la figura 14. Cuanta mayor sea la diferencia de longitudes, más achatada será la elipse hasta llegar a la situación límite en la cual un semirradio vale cero y entonces la elipse se convierte en una recta. En el otro caso, a medida que se aproximen las longitudes de los semirradios, más se parecerá una elipse a una circunferencia.

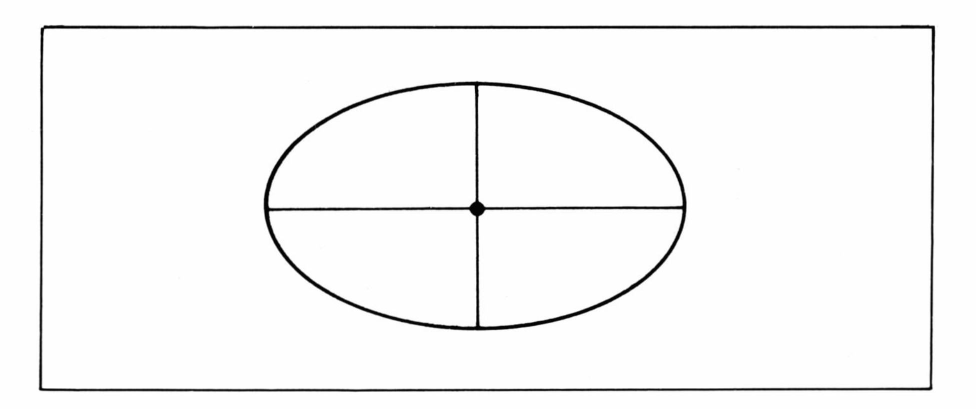


Figura 14 Elipse.

El siguiente programa lo utilizaremos para *dibujar elipses*:

Programa de «Elipses»

```
 10 REM Elipse
 20 LET R1=90: LET R2=20
 30 LET XO=100: LET YO=50
 40 PUNTO(XO+R1,Y)
 50 FOR A=0 TO 6.284 STEP 0.1571
 60   LET X=XO+R1*COS(A)
 70   LET Y=YO+R2*SIN(A)
 80   LINEA(X,Y)
 90   NEXT A
100 STOP
```

En la línea 20 se establecen los valores de los dos semirradios. El resto del programa es muy parecido al utilizado para trazar círculos, excepto en

las líneas 60 y 70 en las que se emplean los valores de R1 y R2 para X y para Y respectivamente.

Puesto que R1 es mayor que R2, la elipse está situada horizontalmente. Si R2 es mayor que R1, la elipse es vertical.

Una espiral se obtiene haciendo girar un punto con radio continuamente creciente. En primer lugar, modificaremos el programa anterior de modo que dé varias vueltas en lugar de una sola. En segundo lugar, haremos que el radio sufra un pequeño incremento continuamente. El programa queda de la forma:

Programa de «Espirales»

```
 10 REM Espiral
 20 LET XO=100: LET YO=50: LET R=20
 30 LET F=5*6.284
 40 LET D=6.284/30
 50 PUNTO(XO+R,Y)
 60 FOR A=0 TO F STEP D
 70   LET X=XO+R*COS(A)
 80   LET Y=YO+R*SIN(A)
 90   LINEA(X,Y)
100   LET R=R+D
110   NEXT A
120 STOP
```

La línea 20 sitúa las coordenadas del centro y define el radio de la primera circunferencia. En la línea siguiente se establece el número de vueltas que dará la espiral, en este caso 5. A continuación (línea 40) se calcula el incremento del bucle de la línea 60. Además, este valor se utiliza también para incrementar el radio (línea 100).

Se pueden obtener figuras bastante originales si, en lugar de dibujar circunferencias, se dibujan polígonos en espiral. Para ello cambiaremos el valor 30 de la línea 40 por un 3, un 4, un 5, etc.

14.4.3 Figuras de Lissajous

Una variante del programa anterior, en el que se incluye un ángulo de desfase, sirve para construir las llamadas *Figuras de Lissajous*. Esta figuras tienen gran importancia en Electrónica.

En nuestro caso el interés se limitará a conocer otro método para trazar figuras con líneas curvas. El programa para dibujarlas es el siguiente:

Programa de «Figuras Lissajous»

```
 10 REM Figuras Lissajous
 20 LET XO=100: LET YO=50
 30 LET XM=50: LET YM=50
 40 LET R=PI/4
 50 LET F=1/3
 60 PUNTO(XO+YM,YO+YM*COS(-R))
 70 FOR A=0 TO 20*PI STEP PI/36
 80   LET X=XO+XM*SIN(A*F)
 90   LET Y=YO+YM*COS(A-R)
100   LINEA(X,Y)
110   NEXT A
120 STOP
```

En la línea 20 establecemos las coordenadas del centro de la figura. En la línea 30, las variables XM e YM son la amplitud máxima horizontal y vertical respectivamente. La variable R de la línea 40 es el ángulo de desfase y la F (línea 50) es un factor que determina la forma. Las líneas comprendidas entre la 60 y la 110 trazan la figura, según un procedimiento parecido al que ya conocemos. Observemos que el bucle va hasta 20 (líneas 70), lo que equivale a 10 vueltas. Después de terminar de dibujar la figura deberá esperar un rato para que el programa termine. En algunos casos hasta dos minutos. Si no quiere esperar, interrumpa con BREAK. Para obtener las diversas figuras hay que variar los valores de F y R.

Para F=1 y R=PI/2 se obtiene una línea recta inclinada.

Para F=1 y R=PI/4 la figura es una elipse. Si R=0, la elipse se convierte en una circunferencia.

Para F=1/3 y R=0 se obtiene una figura parecida a una corona.

Para F=1/3 y R=PI/2 la figura tiene forma de una S horizontal.

Para F=2/3 y R=0 se obtiene una figura en forma de lazo. Si añadimos una pequeña fracción a F, por ejemplo F=2/3+0.005, entonces el lazo no se cierra, adquiriendo la figura una forma de ovillo.



14.5 DIAGRAMAS

Los diagramas se utilizan para la representación visual de los datos, especialmente para el caso de tablas. Los tres tipos más corrientes de diagramas son los de barras, circulares y X–Y. Los dos primeros se utilizan para representar datos estadísticos.

14.5.1 Diagramas de barras

Supongamos que tenemos una serie de datos correspondientes a una magnitud, cuyos valores no pueden ser nunca negativos; por ejemplo, el valor de las ventas de cada mes de una empresa o el número de votos de cada candidato en una elección. Para construir el diagrama, se asigna a cada valor una barra de forma rectangular. La anchura del rectángulo es igual para todos; en cambio, la altura es proporcional al valor del dato. En la figura 15 se ve un ejemplo de este tipo de diagramas.

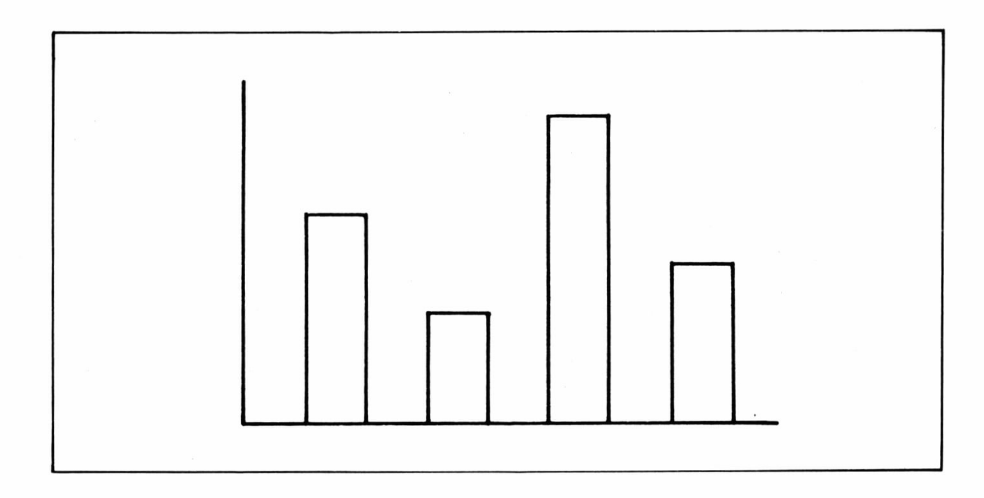
Dado que el valor del dato a representar puede ser mucho mayor que el número de puntos de nuestra pantalla, se hace necesaria una adaptación de los datos. Para ello se escoge el valor máximo de la serie y se le asigna la barra de altura máxima. Entonces todos los demás datos tendrán barras de altura proporcional a su valor numérico y cuya altura será menor que la barra del valor máximo.

El siguiente programa dibuja un diagrama de barras. Los datos a representar están contenidos en una instrucción DATA, pero se pueden cambiar por una instrucción INPUT.

Programa de «Diagrama de barras»

```
 10 REM Diagrama de barras
 20 LET N=5: DIM A(N)
 30 FOR I=1 TO N: READ A(I): NEXT I
 40 LET M=A(1)
 50 FOR I=2 TO N
 60   IF A(I)>M THEN LET M=A(I)
 70   NEXT I
 80 LET E=80/M: LET D=150/(N+1)
 90 GOSUB 300
100 FOR I=1 TO N
110   LET P=I*D: LET L=A(I)*E
120   GOSUB 400
130   NEXT I
140 STOP
150 DATA 10,5,4,20,16
300 REM Ejes
310 PUNTO(20,80)
320 LINEA(20,20)
330 LINEA(170,20)
340 RETURN
400 REM Barra
410 PUNTO(P+20,20)
420 LINEA(P+20,A+20)
430 LINEA(P+30,A+20)
440 LINEA(P+30,20)
450 RETURN
```

Figura 15 Diagrama de barras.



El factor de escala se utiliza para adaptar los datos al tamaño de la pantalla

El programa principal solamente realiza los cálculos. La parte gráfica la realizan las subrutinas 300 y 400.

En la línea 20 se establece un conjunto A de 5 elementos. Posteriormente, este valor puede ser alterado a fin de representar otros conjuntos de datos. La línea 30 es un bucle de lectura de los datos contenidos en la línea 150. Las líneas comprendidas entre 40 y 70 determinan el valor máximo del conjunto A. Este valor se almacena en la variable M. A continuación, en la línea 80 se calculan dos datos importantes. El primero de ellos es el factor de escala, o sea, la relación que hay entre el valor máximo y el número de puntos de la pantalla que ocupa su rectángulo. El segundo valor es la distancia o espaciado entre barras. Se calcula repartiendo equitativamente la anchura total entre todas las barras.

Seguidamente se pasa control a la subrutina 300 donde se dibujan los ejes. Estos ejes no son ejes de coordenadas puesto que sólo tiene graduación el eje vertical. Se utilizan para mejorar la representación visual. Están desplazados 20 puntos hacia arriba y hacia la derecha respecto del punto (0,0).

A partir de aquí se pasa ya a dibujar las barras. La subrutina 400 es la encargada de esta misión. A esta subrutina hay que pasarle dos datos. El primero de ellos es la variable P que contiene la posición (horizontal) donde se dibujará. El segundo, es la variable L que tiene la altura de la barra (medida en número de puntos). Observamos que esta altura se obtiene multiplicando el dato por el factor de escala E (línea 110).

Uno de los perfeccionamientos que se suele añadir a este tipo de diagrama es dibujar algún tipo de rayado en el interior de la barra para hacer que destaque una de las barras respecto a las otras. Otro método sería colorearla, pero éste será un tema que veremos en un capítulo posterior.

Modificaremos el programa de manera que cada dato a representar incluya además el tipo de rayado correspondiente a su rectángulo. Cambiaremos la línea 20 por

```
20 LET N=5: DIM A(N),R(N)
```

y añadiremos las líneas

```
 35 FOR I=1 TO N: READ R(I): NEXT I
160 DATA 2,2,1,2,2
```

En la línea 35 leemos los datos del tipo de rayado que están en la línea 160 y los almacenamos en R. Todas las barras tienen el tipo 2 excepto la tercera que tiene el tipo 1.

A la subrutina 400 hay que añadirle las instrucciones:

```
442 FOR J=20 TO L+20 STEP R(I)
444   PUNTO(P+20,J)
446   LINEA(P+30,J)
448   NEXT J
```

Estas instrucciones forman un bucle que traza rayas horizontales con un espaciado determinado por R(I) a lo largo de la barra. Podemos hacer varias pruebas cambiando los valores de las líneas 150 y 160. Recordemos que si cambiamos el número de datos, que en este momento son 5, hay que cambiar también el valor de N de la línea 20. Además, el número de datos de la línea 150 ha de ser el mismo que el de la línea 160.

14.5.2 Diagramas circulares

Los diagramas circulares se emplean para representar datos del mismo tipo que los diagramas de barras. En esta clase de diagramas se divide un círculo en sectores cuyo ángulo es proporcional al valor del dato. En la figura 16 se muestra un ejemplo.

En un diagrama circular, los valores representados no pueden ser negativos

Para realizar este diagrama, calcularemos en primer lugar el valor de la suma de todos los datos. A este valor le corresponderá la circunferencia completa, o sea, 360 grados (o 2π radianes). Entonces se va repartiendo la circunferencia proporcionalmente a cada valor en relación a la suma total.

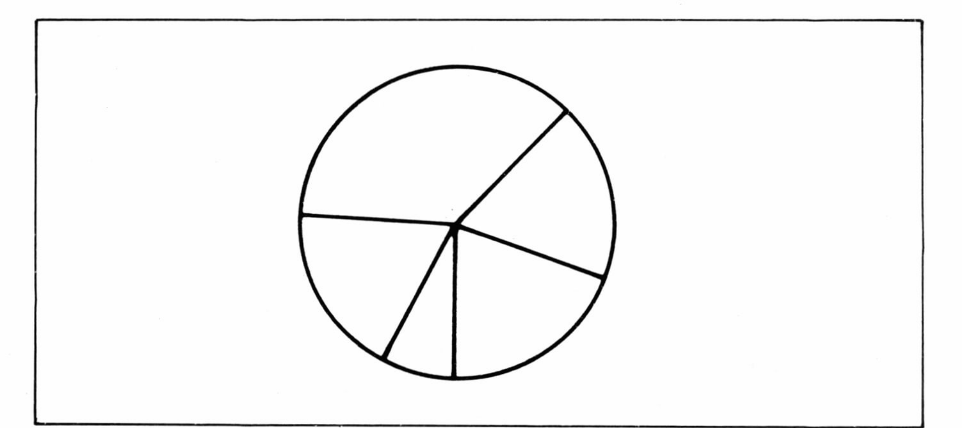


Figura 16 Diagrama circular.

Por ejemplo, supongamos que tenemos los valores 9, 12 y 6. La suma vale 27. Entonces para el valor 9 le asignaremos un sector de 360*9/27=120 grados. Al valor 12 le tocará un ángulo de 360*12/27=160 grados y al valor 6 le corresponden 360*6/27=80 grados. Comprobamos que los tres ángulos suman una circunferencia completa 120+160+80=360 grados.

El siguiente programa sirve para realizar diagramas circulares:

Programa de «Diagramas circulares»

```
 10 REM Diagramas circulares
 20 LET N=5: DIM A(N)
 30 FOR I=1 TO N: READ A(I): NEXT I
 40 LET XO=100: LET YO=50
 50 LET R=40: LET RA=0
 60 LET S=0
 70 FOR I=1 TO N:LET S=S+A(I): NEXT I
 80 FOR I=1 TO N
 90   LET G=6.284*A(I)/S
100   GOSUB 300
110   NEXT I
120 STOP
130 DATA 9,12,6,18,6
300 REM Sectores circulares
310 PUNTO (XO,YO)
320 FOR C=RA TO RA+G STEP 0.1
330   LET X=XO+R*COS(C)
340   LET Y=YO+R*SIN(C)
350   LINEA(X,Y)
360   NEXT C
370 LET RA=RA+G
380 RETURN
```

En las líneas 20 y 30 definimos el conjunto A y lo llenamos con los datos de la línea 130. Observe que son 5 datos y por eso N=5. En la línea 40 establecemos las coordenadas del centro. En la siguiente definimos el valor del radio y una variable auxiliar RA que contendrá el valor del ángulo del último sector dibujado (en este momento es cero). En las líneas 60 y 70 calculamos la suma de los elementos de A y la almacenamos en S. En las líneas comprendidas entre la 80 y la 110 hay un bucle para dibujar N sectores. Notemos que la línea 90 reparte la circunferencia (2π radianes) proporcionalmente a los valores de A. El dibujado de cada sector se realiza en la subrutina 300.

Para dibujar un sector, se coloca un punto en el centro de la circunferencia (línea 310). A continuación, en las líneas comprendidas entre la 320 y la 360 se establece un bucle que dibuja un arco de circunferencia. La primera vez que se efectúe la instrucción 350 se trazará una línea desde el centro hasta el inicio del arco. Este arco se traza desde el valor anterior

contenido en RA hasta RA+G, donde G es el ángulo correspondiente al valor del dato asignado a dicho sector. Finalmente, en la línea 370 se almacena el valor de RA.

Cuando estudiamos el trazado de circunferencias vimos que éstas se construían a base de dibujar un polígono de muchísimos lados. Esto hace que el sector circular de mínimo tamaño que se puede dibujar no debe ser ménor que la longitud del lado del mencionado polígono. Si la circunferencia está formada por un polígono de 120 lados, por ejemplo, entonces el segmento circular mínimo es de 360/120=3 grados. Esto significa que la relación entre la suma de datos y el dato menor ha de ser inferior a 1:120. En efecto, supongamos que los datos a representar en el diagrama son 90, 120, 2 y 60. La suma es 272. Al valor 2 le corresponderían 360*2/272=2.647 grados. En consecuencia, el valor 2 es demasiado pequeño respecto a los demás y no puede ser representado en el diagrama. Lógicamente, si la pantalla tiene mejor resolución, la circunferencia puede estar formada por polígonos de muchos más lados (360 o más). Entonces es posible representar números más pequeños.

Un perfeccionamiento que se suele añadir a estos diagramas es la extracción de un sector a fin de resaltar su valor. Esta extracción se hace de forma análoga a como se separa un trozo de una tarta o pastel. El procedimiento para realizar la extracción no es muy complicado puesto que lo único que hay que variar son las coordenadas del centro del sector circular.

Las variaciones que hay que introducir en el programa anterior son las siguientes:

Se añade la línea 75

```
75 INPUT "SECTOR:",E
```

Esta instrucción pregunta cuál es el sector que queremos extraer. En la subrutina 300 se cambia la línea 310 por

```
310 GOSUB 400
```

La subrutina 400 consta de las siguientes instrucciones:

```
400 REM Centro
410 LET X1=X0: LET Y1=Y0
420 IF I<>E THEN GOTO 450
430   LET X1=X0+R/10*COS(RA+G/2)
440   LET Y1=Y0+R/10*SIN(RA+G/2)
450 PUNTO(X1,Y1)
460 RETURN
```

En las líneas 330 y 340 cambiaremos XO e YO por X1 e Y1 respectivamente, quedando

```
330 LET X=X1+R*COS(C)
340 LET Y=Y1+R*SIN(C)
```

Finalmente añadiremos la línea

```
365 LINEA X1,Y1
```

La extracción de un sector nos permite resaltar un valor determinado



Esta última línea es necesaria ya que anteriormente los sectores circulares compartían los radios con los sectores adyacentes. De esta manera bastaba con dibujar uno solo de ellos. Ahora esta condición no se cumple puesto que un sector puede ser extraído y en consecuencia necesita los dos radios que lo delimitan.

14.5.3 Diagramas X–Y

Los diagramas X–Y se utilizan en dos casos. En el primer caso se trata de trazar la gráfica de una función y=f(x), como por ejemplo el seno o la raíz cuadrada. En el segundo caso se trata de representar parejas de datos que corresponden a las coordenadas de un punto. Ejemplos de este último caso pueden ser el índice de la cotización de la bolsa o la evolución de la fiebre de un enfermo.

Para el primer caso trazaremos unos ejes de coordenadas cuyo origen esté situado en el centro de la pantalla. Si la anchura de la pantalla es 200, entonces se considera que la posición 100 corresponde a X=0. A partir de aquí se calculan los valores de Y para los 200 valores de X.

El siguiente programa lo utilizaremos para *graficar funciones.*

Programa de «Funciones»

```
 10 REM Graficas funciones
 20 INPUT "INTERVALO:",S
 30 PUNTO(100,100): LINEA(100,0)
 40 PUNTO(0,50): LINEA(200,50)
 50 DEF FNA(X)=40*SIN(X)
 60 LET A=0
 70 FOR X=0 TO 200
 80   LET Z=S*(X-100)/100
 90   LET Y=FNA(Z)
100   IF ABS(Y)>50 THEN LET A=0: GOTO 140
110   LET Y=Y+50
120   IF A=0 THEN PUNTO(X,Y): LET A=1:GOTO140
130   LINEA(X,Y)
140   NEXT X
150 STOP
```

En la línea 10 se pide el intervalo de valores de X entre los cuales se dibujará la gráfica. Las líneas 30 y 40 trazan los ejes. En la línea 50 se define la función a graficar. La función escogida (40sen x) se dibuja desde –5 hasta +5. La variable A de la línea 60 sólo tiene dos valores posibles. Si vale cero significa que se sigue dibujando a partir del punto anterior. Este control es importante, puesto que según sean los valores de Y, algún punto puede salir fuera del gráfico para ciertos valores de X. Entonces cuando vuelvan a caer dentro del gráfico hay que empezar de nuevo, colocando un punto.

La línea 70 establece el bucle para calcular 200 puntos. En la línea 80 se calcula el valor real de la ordenada según el intervalo y según el valor de X. En la línea 90 se calcula Y según la función definida por el usuario. A continuación (línea 100) comprobamos si el valor de Y sale fuera de margen. Si es así, se pone A igual a cero y se salta al final del bucle. En caso contrario se le añade el desplazamiento del eje y se comprueba (línea 120) si A valía cero. Si es así, se coloca un punto, se pone un 1 en A y se va al final del bucle. Si A no era cero, entonces se traza una línea (línea 130).

Probaremos este programa para S=8, S=16, etc. Podemos hacer que el programa vaya más rápido, sin perder demasiada precisión, cambiando la línea 70 por

```
70 FOR I = 0 TO 200 STEP 2
```

Otras funciones que podemos graficar son

```
50 DEF FNA(X) = X*X-40      con S=7
50 DEF FNA(X)=16*EXP(-(X*X-1)) con S=4
50 DEF FNA(X)=10*TAN(X)    con S=10
```

Para los diagramas X–Y donde se *grafican puntos concretos* utilizaremos el siguiente programa:

Programa para dibujar «Puntos concretos»

```
 10 REM Graficos X-Y
 20 LET N=5: DIM A(N),B(N)
 30 FOR I=1 TO N
 40   READ A(I),B(I)
 50   NEXT I
 60 LET XA=A(1): LET XI=A(1)
 70 LET YA=B(1): LET YI=B(1)
 80 FOR I=2 TO N
 90   IF XA<A(I) THEN LET XA=A(I)
100   IF XI>A(I) THEN LET XI=A(I)
110   IF YA<B(I) THEN LET YA=B(I)
120   IF YI>B(I) THEN LET YI=B(I)
```

```
130    NEXT I
140 LET EX=(XA-XI)/200: LET EY=(YA-YI)/100
150 LET X=A(1)/EX: LET Y=B(1)/EY
160 PUNTO(X,Y)
170 FOR I=2 TO N
180   LET X=A(I)/EX: LET Y=B(I)/EY
190   LINEA(X,Y)
200   NEXT I
210 STOP
220 DATA 1,20,5,11,10,10
230 DATA 20,30,25,20
```

El diagrama X–Y necesita un factor de escala para cada eje

En la línea 20 se definen dos conjuntos A y B que contendrán las coordenadas de los puntos. El conjunto A tendrá la coordenada X y el conjunto B la coordenada Y. Las instrucciones 30, 40 y 50 obtienen las coordenadas a partir de los datos de las líneas 220 y 230. Las líneas comprendidas entre la 60 y la 130 determinan los valores máximos y mínimos de A y B. Las variables XA y XI contienen el máximo y el mínimo respectivamente de A (coordenada X). Las variables YA e YI contienen el máximo y el mínimo respectivamente de B (coordenada Y).

En la línea 140 se aprovechan los valores anteriores para calcular los dos factores de escala EX y EY. Estos factores de escala sirven para adaptar los datos a las dimensiones de la pantalla que son siempre fijas. De este modo podremos representar sin problemas, datos que varíen, por ejemplo, entre 1 y 10, o bien entre 1 millón y 10 millones. En ambos casos el factor de escala los transformará y obtendremos valores representables en la pantalla. Como es lógico, se calcula un factor de escala para cada eje.

En las líneas 140 y 150 se calculan las coordenadas reales del primer punto y se dibuja éste. A continuación se establece un bucle (líneas 170 a 200) donde se trazan las líneas que unirán los diversos puntos formando el gráfico.

En las instrucciones DATA hay 10 valores que forman 5 parejas de coordenadas. Si queremos representar otros puntos, deberemos dar sus coordenadas de la forma indicada y cambiar el valor de N (línea 20) si el número de puntos a representar es distinto de 5.



14.6 SUPERPOSICIONES DE LINEAS

Como sabemos, el estado de un pixel (activado o desactivado) se memoriza en un bit de la memoria central. Cuando este bit se pasa de 0 a 1, el pixel asociado se activa. ¿Qué ocurre, sin embargo, cuando se intenta activar un pixel que ya está activado? Este hecho, se produce por ejemplo, cuando se cruzan dos líneas. En el punto de cruce, el pixel se ha activado dos veces. Probablemente, al realizar los gráficos anteriores ya hemos observado este hecho y nuestra experiencia nos indica que no pasa nada. Una vez el pixel está activado, sucesivas activaciones no tienen efecto. Ahora bien, ¿Podría ocurrir esto de otra manera? La respuesta es sí.

En la pantalla coexisten dos colores: el de fondo o papel y el de las líneas o tinta

En primer lugar debemos definir algunos conceptos básicos. Cuando se traza una línea, ésta se dibuja utilizando el color inverso al de la pantalla. Es decir, que si la pantalla es negra, la línea se dibujará con fondo blanco y viceversa. Como ya dijimos al principio, en este capítulo sólo trataremos de los gráficos en blanco y negro. Por tanto, sólo tendremos estos dos colores. El color de la pantalla se denomina color de fondo («background» en inglés) o también color del papel. El color de las líneas se denomina a veces color de la «tinta» por analogía con los dibujos sobre papel. Pues bien, tanto el color de fondo como el color de la tinta se pueden invertir. Es posible entonces pasar de tener una pantalla blanca y líneas negras a tener un fondo de color negro con rayas blancas. Si sólo se invierte uno de los colores, entonces se dibuja de forma «invisible», es decir, blanco sobre blanco, o negro sobre negro. Este proceso parece totalmente inútil, pero en cambio es de gran importancia, pues es el que nos permite borrar líneas ya trazadas, como veremos más adelante.

Otro concepto importante es el de la superposición. Por el momento, la activación o desactivación de un pixel es independiente de su estado anterior. Cuando trazamos una línea se activan los pixels estuvieran o no activados previamente. Asimismo, al trazar una línea en modo inverso, los pixels pasan al estado de desactivación sin tener en cuenta su situación anterior. Es posible activar o desactivar un pixel según su estado anterior. Esta operación se denomina trazado con superposición. Cuando se emplea este sistema, la línea se traza de la siguiente manera: si el pixel está desactivado (caso normal) se activa éste. Por el contrario, si ya está activado, pasa al estado de desactivación. De esta forma, si trazamos una línea en modo superposición que se cruce con otra, la línea se dibujará normalmente excepto en el punto de cruce que quedará desactivado.

A nuestras dos instrucciones básicas PUNTO y LINEA les añadiremos las opciones de trazado inverso y con superposición. El trazado inverso se indicará

```
PUNTO(X,Y),I
LINEA(X,Y),I
```

El trazado con superposición lo indicaremos

```
PUNTO(X,Y),S
LINEA(X,Y),S
```

Naturalmente, en nuestro ordenador se indicará de forma distinta, cosa que ya sabemos por estar especificada en el capítulo de «Prácticas con el microordenador».

14.6.1 Borrado de líneas

Aplicaremos los conceptos anteriores para borrar líneas. Escribamos el siguiente programa.

```
10 PUNTO(1,1)
20 LINEA(70,30)
30 INPUT A$
40 PUNTO(1,1),I
50 LINEA(70,30),I
```

El trazado en modo inverso sirve para borrar

Las dos primeras instrucciones trazan una línea. A continuación se utiliza una instrucción INPUT para que el programa se detenga hasta que el operador teclee cualquier cosa. Inmediatamente el programa borra la línea trazada. Esto lo realizamos dibujando de nuevo la misma línea en modo inverso.

Al borrar la línea debemos seguir la misma dirección en que se dibujó inicialmente. Si modificamos las dos últimas instrucciones de la forma

```
40 PUNTO(70,30),I
50 LINEA(1,1),I
```

veremos que la línea se borra, pero quedan algunos puntos sueltos sin borrar. Esto se debe a la forma de trazado de una línea descrita en la figura 6. Si se traza al revés, a causa de los errores de redondeo al calcular la pendiente de la recta, algunos pixels que son afectados en una dirección, no lo son en la dirección contraria, dando lugar a un borrado imperfecto. Por tanto, el borrado de una recta se realizará siempre siguiendo la misma dirección con la que se dibujó inicialmente.



14.6.2 Dibujo interactivo

Aplicaremos estos conocimientos recién adquiridos para construir un programa que nos permitirá elaborar dibujos interactivamente, o dicho en otras palabras, utilizar la pantalla como papel de dibujo.

El funcionamiento será el siguiente: En la pantalla aparecerá un punto que podremos desplazar en las cuatro direcciones (arriba, abajo, izquierda y derecha). Mediante algunas teclas determinadas, le podremos indicar que una dos puntos con una línea o que dibuje un círculo, etc.

Desplazar un punto por la pantalla se basa en el conocido sistema de dibujar un punto al lado y borrar el anterior. Como esta operación se realiza muy rápidamente, se obtiene la sensación de movimiento.

La función INKEY$ nos permite dar órdenes sin pulsar la tecla de fin de línea

Para determinar qué tecla ha apretado el operador, utilizaremos la función INKEY$. Este programa es un poco largo pero es bastante interesante, pues incluye varios conceptos importantes. En primer lugar constituye un buen repaso de las instrucciones gráficas del ordenador. Por otra parte, utilizaremos la técnica interactiva, es decir, que el programa sólo actúa bajo órdenes directas del operador. Finalmente es una muestra patente de cómo debe utilizarse la técnica de modularización para construir un programa. En definitiva es una buena práctica, razón por la que en esta unidad no le in-

cluiremos ninguna otra práctica especial. El *primer paso* consiste en definir las variables que necesitamos.

XM: Valor más a la derecha que puede alcanzar el punto.

YM: Valor más alto que puede alcanzar el punto.

X: Coordenada *x* del punto. El valor de X estará comprendido entre 0 y XM.

Y: Coordenada *y* del punto. El valor de Y estará comprendido siempre entre 0 e YM.

T$: Variable que contendrá la tecla pulsada por el operador.

El *segundo paso* consiste en definir las operaciones a realizar. Inmediatamente vemos que hay cuatro operaciones básicas que son los movimientos en las cuatro direcciones. Estas cuatro operaciones las realizarán las subrutinas 200, 300, 400 y 500. Otras dos operaciones a efectuar son el marcado del inicio y del final de una línea, de las cuales se encargarán las subrutinas 600 y 700.

Para indicar al programa que mueva el punto en la dirección adecuada, utilizaremos las teclas de control del cursor (flechas) cuyos códigos supondremos que son 8, 9, 10 y 11, que corresponden al desplazamiento hacia la izquierda, hacia la derecha, hacia abajo y hacia arriba, respectivamente. Es posible que alguno de estos códigos sea distinto en nuestro ordenador. En el capítulo de prácticas con el microordenador se indican los códigos correspondientes en nuestra máquina. Para indicar que queremos marcar un punto emplearemos la letra P y una L para trazar una línea. Finalmente, reservaremos la tecla F para detener el programa.

El programa principal es el siguiente:

Programa de «dibujo interactivo»

```
 10 REM Dibujo interactivo
 20 LET XM=100: LET YM=50: LET P=0
 30 LET X=XM/2: LET Y=YM/2: GOSUB 200
 40 LET T$=INKEY$
 50 IF T$=CHR$(9) THEN GOSUB 200: GOTO 40
 60 IF T$=CHR$(8) THEN GOSUB 300: GOTO 40
 70 IF T$=CHR$(11) THEN GOSUB 400: GOTO 40
 80 IF T$=CHR$(10) THEN GOSUB 500: GOTO 40
 90 IF T$="P" OR T$="p" THEN GOSUB 600: GOTO40
100 IF T$="L" OR T$="l" THEN GOSUB 700: GOTO40
180 IF T$="F" OR T$="f" THEN STOP
190 GOTO 40
```

El movimiento del punto se efectúa borrando su

En la línea 20 establecemos las medidas de la pantalla en la que se moverá el punto. La variable P es una variable auxiliar que si vale cero

posición anterior y dibujándolo otra vez en la nueva posición

indica que nos movemos sin dibujar y si vale 1 indica que dibujamos. En la siguiente línea hacemos que los valores de las coordenadas del punto sean las del centro de la pantalla. A continuación lo enviamos a la subrutina 200 para hacerlo visible.

En la línea 40 obtenemos el valor de la tecla pulsada mediante la función INKEY$. Las líneas comprendidas entre la 50 y la 180 sirven para pasar control a la subrutina correspondiente a la tecla pulsada. Fijémonos que las flechas no son caracteres imprimibles y por tanto es obligatorio usar la función CHR$. Para las letras P, L y F se tiene en cuenta la posibilidad de que sean mayúsculas o minúsculas haciendo una pregunta doble utilizando el operador booleano OR (Si T$ es igual a «P» *ó* T$ es igual a «p»). Después de realizar la acción o bien si se pulsa una tecla no admitida, se pasa control de nuevo a la línea 40. Entre la línea 100 y la 180 se ha dejado espacio para introducir nuevas operaciones.

Una vez elaborado el programa principal hay que añadirle los diversos módulos que realizan las operaciones. Estos módulos se describen a continuación:

Listado de la subrutina para realizar el movimiento hacia la derecha.

```
200 IF X=XM THEN RETURN
210 IF P=0 THEN PUNTO(X,Y),I
220 LET X=X+1: LET P=0
230 PUNTO(X,Y)
240 RETURN
```

Subrutina para efectuar el movimiento hacia la izquierda.

```
300 IF X=0 THEN RETURN
310 IF P=0 THEN PUNTO(X,Y),I
320 LET X=X-1: LET P=0
330 PUNTO(X,Y)
340 RETURN
```

Subrutina para realizar el movimiento hacia arriba.

```
400 IF Y=YM THEN RETURN
410 IF P=0 THEN PUNTO(X,Y),I
420 LET Y=Y+1: LET P=0
430 PUNTO(X,Y)
440 RETURN
```

Y, finalmente, la subrutina para realizar el movimiento hacia abajo.

```
500 IF Y=0 THEN RETURN
```

```
510 IF P=0 THEN PUNTO(X,Y),I
520 LET Y=Y-1: LET P=0
530 PUNTO(X,Y)
540 RETURN
```

Estas cuatro subrutinas tienen una estructura análoga, por tanto describiremos únicamente la primera de ellas. En primer lugar (línea 200) se pregunta si ya hemos llegado al tope de la pantalla (XM). Si es así, volvemos al programa principal sin hacer nada. En caso contrario hay que proceder a mover el punto. Previamente averiguamos si nos movemos dibujando o no (variable P). Si sólo hay que mover el punto, entonces lo borramos de la posición actual (línea 210). A continuación incrementamos la coordenada X (línea 220) y lo dibujamos de nuevo (línea 230). Las otras tres subrutinas tienen un comportamiento idéntico y únicamente varían el límite y el incremento (o decremento) de la variable.

De momento ya podemos mover el punto. Falta ahora el trazado de líneas. De esto se encargan las subrutinas 600 y 700.

```
600 LET P=1
610 LET XA=X: LET YA=Y
620 RETURN
```

```
700 LET P=1
710 PUNTO(XA,YA)
720 LINEA(X,Y)
730 RETURN
```

En la línea 600 colocamos un 1 en P que indica que vamos a marcar un punto. A continuación almacenamos en XA e YA las coordenadas de este punto. En la subrutina 700 hacemos la misma operación con la variable P y a continuación trazamos la recta desde (XA, YA) hasta (X, Y).

Ya podemos operar con el programa. Si no hemos cometido ningún error al teclearlo, al hacer un RUN aparecerá en pantalla un punto en el centro de la misma. Pulsando las teclas de las flechas veremos que el punto se desplaza en la dirección indicada. También comprobamos que al llegar al límite de la pantalla, el movimiento se detiene.

Veamos ahora cómo se traza una línea. Situaremos el punto en la posición en la cual debe empezar la línea y pulsaremos una P. Al desplazar el punto veremos que en la posición indicada permanece un punto que actúa de punto actual. Colocaremos ahora el punto móvil en la posición donde debe finalizar la línea. Al pulsar una L, las dos posiciones quedan enlazadas por una recta. En todo caso, no pulse nunca *L* sin haber pulsado *P*, pues no tendría sentido y le daría error. Además, la última posición pasa a ser ahora la posición actual y, por tanto, si desplazamos el punto móvil y pulsamos otra L, se traza una nueva recta desde donde terminó la anterior hasta la nueva posición. Si queremos trazar líneas que no estén unidas a

las anteriores, utilizaremos la tecla P para establecer un nuevo inicio de línea.

Para finalizar, pulsaremos la tecla F.

Este programa está preparado para que se puedan añadir fácilmente nuevas operaciones, como por ejemplo, trazado de círculos, triángulos o cualquier otra. El procedimiento para añadir una nueva operación es siempre el mismo y lo describiremos con un ejemplo. El ejemplo escogido consistirá en el proceso de borrado de líneas. Veamos paso a paso cómo se añade esta operación al programa principal.

En primer lugar hay que hacer un planteo claro del funcionamiento de la operación, es decir, realizar el diseño. En este caso decidimos que el funcionamiento sea análogo al trazado de una línea, excepto que este trazado se efectuará en modo inverso.

A continuación, hay que construir el módulo que realice la operación, formado por una o varias subrutinas. En este caso, consistirá en una simple adaptación de la subrutina empleada en el trazado de rectas. El listado es:

```
800 PUNTO(XA,YA),I
810 LINEA(X,Y),I
820 RETURN
```

Se ha elegido el número 800 como inicio de la numeración, puesto que la última subrutina tenía el 700. Sin embargo, serviría cualquier otra numeración. Como se aprecia, lo único que hace esta subrutina es trazar una recta en modo inverso. Además, en este caso no hay que colocar un 1 en la variable P, puesto que precisamente estamos borrando. Recordemos que si P vale 1, las subrutinas que realizan los movimientos interpretarán que hay que dejar dibujado un punto en la pantalla.

Seguidamente, decidimos la tecla que al ser pulsada por el operador desencadenará la acción. Elegimos la letra B (de Borrar). Entonces en el programa principal se inserta la siguiente línea:

```
110IF T$="B" OR T$="b" THEN GOSUB 800:GO TO40
```

Puesto que en su momento dejamos un hueco en la numeración (de la línea 100 pasamos a la 180), la inserción es sumamente sencilla.

Finalmente, una vez tecleadas todas estas instrucciones, viene la fase de prueba y puesta a punto. Para borrar una línea, colocaremos el punto móvil sobre el inicio de una línea y pulsaremos una P. A continuación colocaremos el punto sobre el otro extremo de la recta y entonces pulsaremos una B. El programa trazará una recta en modo inverso que eliminará la línea inicial. Recordemos que para lograr un borrado perfecto, hay que trazar la línea inversa en la misma dirección que la línea original.

RESUMEN

Los polígonos regulares se construirán aprovechando la propiedad de que pueden inscribirse dentro de un círculo. Dibujando un polígono regular con un número elevado de lados obtendremos una figura equivalente a un círculo. En ambos casos emplearemos la función COS para calcular la coordenada X y la función SIN para calcular la coordenada Y.

Para obtener elipses, se modifica el programa de modo que existan dos semiradios en lugar de uno solo, como en el caso de la circunferencia. Una línea espiral se elaborará haciendo girar un punto cuya distancia al centro sea creciente.

Para la representación visual de los datos se emplean principalmente tres tipos de diagramas; de barras, circulares y X–Y. Los de barras se basan en asignar a cada dato una barra cuya altura sea proporcional al valor del dato. Todas las barras tienen la misma anchura. Los diagramas circulares se basan en dividir los 360 grados de la circunferencia en sectores cuyo ángulo es proporcional al valor del dato representado. Estos dos tipos de diagramas se utilizan principalmente para representar datos de tipo estadístico. El diagrama X–Y se emplea para representar gráficas de funciones o para representar tablas de valores. Estas tablas suelen contener los diversos valores que toma una magnitud, frecuentemente, respecto al tiempo. Tanto en los diagramas de barras como en los diagramas X–Y hay que utilizar el factor de escala para adaptar los datos a las dimensiones del gráfico.

EJERCICIOS DE AUTOCOMPROBACION

Completar las frases siguientes:

31. En pantalla, un polígono de muchos lados es equivalente a un ....

32. En una elipse existen dos .... de longitud distinta.

33. Un punto que gire con radio .... traza una línea espiral.

34. El factor de .... se utiliza para adaptar los datos a las dimensiones del gráfico.

35. El .... de un sector de un diagrama circular es proporcional al valor del dato.

36. Al trazar una circunferencia se empleará la función trigonométrica .... para calcular la coordenada Y.

37. Las cotizaciones de una divisa extranjera a lo largo de un mes se representarían en un diagrama ....

38. Al trazar una línea en modo .... los pixels se desactivan.

39. Una línea trazada en modo superposición, cuando intercepte pixels activados, ocasionará su ....

40. El trazado en modo inverso lo utilizaremos para .... líneas.

Rodee con una circunferencia la letra que corresponde a la respuesta que considere correcta.

41. Para elaborar un programa que dibuje polígonos regulares nos basamos en el hecho de que

a) Todos sus vértices tienen ángulos inferiores a 90 grados.

b) Todo polígono regular se puede inscribir dentro de un círculo.

c) El número de pixels es limitado.

d) El número de lados es múltiplo de 360.

42. En un diagrama de barras, la altura de una barra es

a) Igual al valor máximo.

b) Proporcional a la distancia al centro de coordenadas.

c) Equivalente a la suma de todos los valores.

d) Proporcional al valor representado.

43. Para extraer un sector en un diagrama circular, el programa debe:

a) Avanzar 60 grados.

b) Incrementar el valor de la coordenada X.

c) Avanzar un número de grados igual a la anchura del sector.

d) Variar momentáneamente las coordenadas del centro.

44. Cuando se utiliza el modo superposición, la activación del pixel:

a) Es inmediata.

b) Depende de su estado anterior.

c) No tiene lugar.

d) Se realiza de forma invisible.

45. En un programa que permita realizar gráficos interactivamente, lo primero que debe controlarse es:

a) Que durante el movimiento, el punto no se salga de los límites de la pantalla.

b) Que las líneas rectas sean horizontales o verticales.

c) Que los círculos estén centrados.

d) Que la coordenada X avance más rápidamente que la Y.

Según lo estudiado hasta ahora en la lección responda si son correctas (S) o incorrectas (N) las afirmaciones siguientes:

46. En un diagrama de barras, la anchura debe ser igual para todas las barras. S N

47. Los valores representados en un diagrama circular nunca pueden ser negativos. S N

48. La variación diaria del índice de humedad ambiental se representaría en un diagrama circular. S N

49. El número más pequeño que se puede representar en un diagrama circular es igual a 360 multiplicado por la suma del resto de valores. S N

50. En un diagrama X–Y hay que utilizar dos factores de escala. S N

Capítulo 15
• Dibujos bidimensionales y tridimensionales

ESQUEMA DE CONTENIDO

15.0 OBJETIVOS

En este capítulo nos proponemos completar los conocimientos sobre gráficos y sonido. Sin embargo, quedarán algunos pequeños puntos concretos que reservaremos para cuando sepamos acceder al código máquina de nuestro ordenador.

En primer lugar terminaremos el estudio de los gráficos bidimensionales. Aprenderemos las operaciones básicas que se pueden realizar sobre las figuras. Estas operaciones son la traslación para mover figuras por la pantalla, el escalado para cambiar el tamaño de la figura y la rotación para modificar la orientación de la figura.

Seguidamente pasaremos al estudio de figuras tridimensionales. Veremos la técnica de la perspectiva y los procedimientos para representar superficies como son el sombreado y la retícula. Este último método nos permitirá obtener figuras con apariencia sólida o transparente. Para finalizar los gráficos tridimensionales, estudiaremos la manera de representar funciones X-Y-Z.

Otro tema muy importante en los gráficos, es el color. Por tanto estudiaremos la teoría básica y aprenderemos los conceptos fundamentales sobre tono, saturación y brillo. Asimismo, veremos lo que son mezclas aditivas y sustractivas y el concepto de colores primarios. Aquí revisaremos también la relación entre lo que está representado en pantalla y el contenido en memoria.

Debido a la gran importancia de este tema, veremos varias aplicaciones que incluirán mejoras en los diagramas vistos en el capítulo anterior y representación de objetos sólidos iluminados.

Finalmente, aprenderemos cómo memorizar una pantalla total o parcialmente y cómo averiguar el estado completo de un pixel de la pantalla.

Respecto a la utilización de sonido por ordenador, veremos la orden fundamental que nos permite elegir el tono y la duración de un sonido determinado. Mediante la combinación adecuada de varias instrucciones de este tipo veremos cómo reproducir una composición musical.



15.1 OPERACIONES BIDIMENSIONALES

En esta segunda parte dedicada a gráficos haremos un uso constante de los conceptos fundamentales sobre gráficos explicados en el capítulo anterior. Convendrá, pues, realizar un repaso antes de empezar.

Mantendremos también, las mismas convenciones que en el capítulo anterior, tanto respecto al tamaño de pantalla como a las instrucciones no estándar del BASIC.

La ventaja de emplear un ordenador para efectuar dibujos, consiste en que podemos programarlo para que mueva la figura sobre la pantalla, realice un escalado, efectúe rotaciones, etc. Para realizar estas operaciones necesitamos construir, en primer lugar, un programa que sepa dibujar cualquier tipo de figura.

15.1.1 Memorización coordenadas

Memorización de las coordenadas en figuras irregulares

En programas anteriores hemos visto la manera de dibujar figuras planas de tipo regular, es decir, aquellas cuyo perímetro puede calcularse mediante una fórmula matemática. Dentro de este tipo de figuras podemos incluir a los círculos, elipses, polígonos regulares, etc. Ahora bien, en la práctica nos encontramos a menudo con figuras de tipo irregular. ¿Qué hacer en este caso? La solución consiste en memorizar las coordenadas de los vértices. Entonces elaboraremos un programa para que, dado un conjunto de coordenadas de puntos, los sitúe en la pantalla y los una con líneas rectas.

El procedimiento anterior es válido siempre. Si la figura tiene alguna línea curva, se descompondrá ésta en una serie de pequeñas líneas rectas. Recordemos que para trazar una circunferencia aplicábamos este principio, ya que en realidad dibujábamos un polígono con muchos lados. Este hecho implica que, en principio, para memorizar una figura que contenga líneas curvas, necesitaremos memorizar un conjunto mucho mayor de coordenadas que si la figura está formada exclusivamente por líneas rectas.

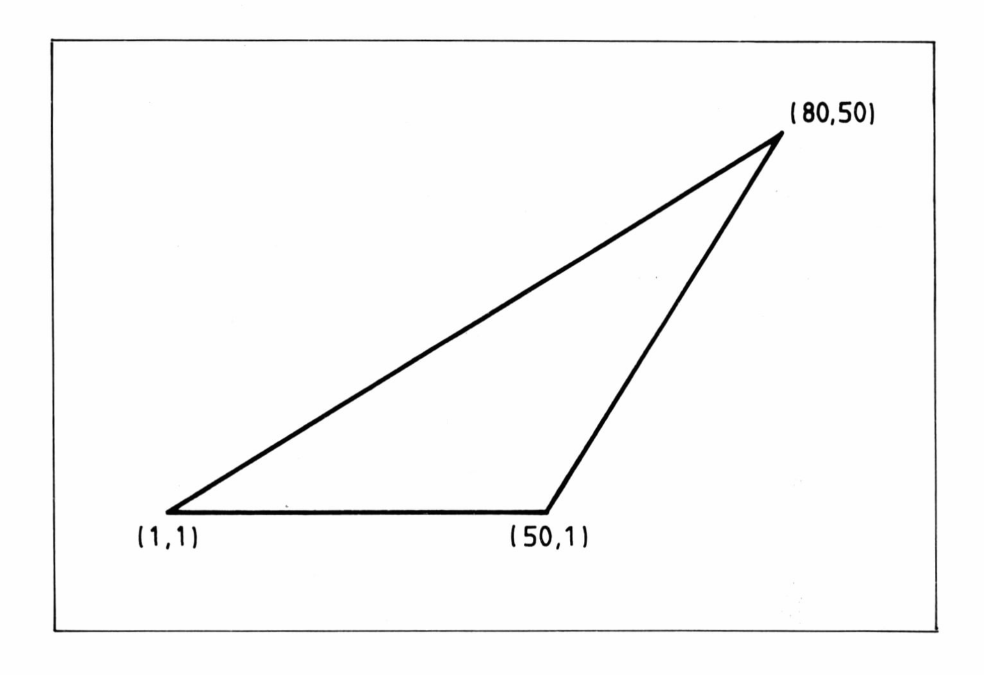
Tomemos el triángulo de la figura 1. Las coordenadas de los tres vértices son (1,1), (50,1) y (80,50). Para almacenar la figura (un triángulo en este caso) basta con memorizar estos tres pares de números, puesto que este triángulo es el único que tiene precisamente este conjunto de coordenadas. Lo mismo se aplica para figuras con un número más elevado de vértices, para las cuales se definirá un conjunto mayor de coordenadas.

Construyamos pues, un programa para realizar estos dibujos. La operación de trazado de líneas de unión entre vértices la construiremos mediante una subrutina. De esta forma, podremos utilizarla con facilidad en programas posteriores. El listado del programa es el siguiente:

Programa de «dibujo de figuras»

```
 10 REM Dibujo figuras
 20 LET N=4: DIM A(N),B(N)
 30 FOR I=1 TO N
 40   READ A(I),B(I)
 50   NEXT I
 60 GOSUB 500
 70 STOP
 80 DATA 1,1,50,1,80,50,1,1
500 REM Trazado figuras
510 LET X=A(1):LET Y=B(1): PUNTO(X,Y)
520 FOR I=2 TO N
530   LET X=A(I):LET Y=B(I)
540   LINEA(X,Y)
550   NEXT I
560 RETURN
```

Figura 1. Coordenadas de los vértices de un triángulo.



En primer lugar, el programa define dos conjuntos, A y B, en donde se almacenarán los valores de las coordenadas. El tamaño o dimensión de estos conjuntos se establece mediante la variable N. De esta manera, con sólo cambiar el valor de N, el programa sirve para representar cualquier figura. En el conjunto A se almacenarán las coordenadas X de cada vértice y en el conjunto B, las coordenadas Y. Notemos que N vale 4 mientras que la figura sólo tiene 3 vértices. El cuarto punto es necesario para cerrar la figura. Por esta razón, sus coordenadas coinciden con las del primer vértice.

Definición del punto actual

Las líneas comprendidas entre la 30 y 50 forman el conocido bucle de llenado de un conjunto dimensionado. Seguidamente se pasa control a la subrutina 500, que es donde se efectúa realmente el trazado de la figura. La primera operación que realiza esa subrutina es colocar el punto o vértice inicial de la figura. A continuación se van trazando líneas desde un vértice a otro. Podríamos preguntarnos la razón por la cual el primer punto se dibuja de forma distinta al resto. La causa es que el trazado de una línea se realiza desde el *punto actual* hasta las coordenadas del punto final. Por tanto, cuando se dibuja la primera línea es necesario que ya esté definido el punto actual. Por esta razón, el primer punto se dibuja aparte de los demás.

En la línea 80 se colocan los valores de las coordenadas de los vértices de la figura. Estas coordenadas se almacenan por pares, es decir, coordenadas X e Y del primer punto, coordenadas X e Y del segundo punto y así sucesivamente.

Este programa sirve para cualquier figura, sea cual sea su número de lados. Podemos probarlo, añadiendo nuevos pares de coordenadas a la instrucción DATA, o incluso añadiendo nuevas instrucciones DATA si hace falta. Como es lógico, el valor de N se retocará oportunamente para que corresponda al número de vértices. Es importante comprobar, antes de ejecutar el programa, que el número total de datos contenidos en las instrucciones DATA sea exactamente el doble de N. Si no es así, es que hay un error en la lista de coordenadas.



15.1.2 Traslación

Traslación de figuras

La operación más sencilla que se puede realizar sobre una figura bidimensional es desplazarla sobre la pantalla. Esta operación recibe el nombre de traslación. Como ya sabemos de programas anteriores, para desplazar un punto o un carácter en la pantalla, se borra la posición donde se encuentra en ese momento y seguidamente se dibuja de nuevo al lado. El proceso es parecido cuando se trata de trasladar una figura, con la salvedad de que es un poco más laborioso, ya que ahora no se trata de un punto o carácter aislado, sino de un conjunto de líneas.

En el programa que nos ocupa, tenemos una subrutina que dibuja la figura. Según lo que acabamos de explicar, necesitaremos unas instrucciones para borrarla. Como de costumbre, las incluiremos también en una subrutina. Para empezar, elaboraremos un programa que realice únicamente desplazamientos horizontales. El listado es el siguiente:

Programa de «Traslación lateral»

```
 10 REM Traslacion lateral
 20 LET N=13: DIM A(N),B(N)
 30 FOR I=1 TO N
 40   READ A(I),B(I)
 50   NEXT I
 60 GOSUB 500
 70 INPUT "DESPLAZAMIENTO:";D
 80 GOSUB 600
 90 FOR I=1 TO N:LET A(I)=A(I)+D: NEXT I
100 GOSUB 500
110 STOP
120 DATA 10,50, 30,50, 30,70, 50,70
130 DATA 50,50, 70,50, 70,30, 50,30
140 DATA 50,10, 30,10, 30,30, 10,30, 10,50
500 REM Trazado lineas
510 LET X=A(1):LET  Y=B(1): PUNTO(X,Y)
520 FOR I=2 TO N
530   LET X=A(I):LET Y=B(I)
540   LINEA(X,Y)
550   NEXT I
560 RETURN
600 REM Borrado lineas
610 LET X=A(1):LET Y=B(1): PUNTO(X,Y),I
620 FOR I=2 TO N
630   LET X=A(I):LET Y=B(I)
640   LINEA(X,Y),I
650   NEXT I
660 RETURN
```

Las primeras instrucciones hasta la línea 60, coinciden con el programa inicial. En este caso se ha escogido una figura más compleja (una cruz) con mayor número de vértices y, por tanto, se ha aumentado el valor de N. La línea 60 pasa control a la subrutina 500 donde se dibuja la figura. A continuación, el programa pide el valor del desplazamiento (variable D). Es decir, cuánto la quiere usted desplazar. Déle un número, por ejemplo: 50. A partir de ahora empieza la fase de traslado.

En primer lugar, pasa control a la subrutina 600 para borrar la figura. Inmediatamente se procede a incrementar el valor de las coordenadas X de los vértices, contenidas en el conjunto A. Este incremento se realiza mediante un bucle que suma el valor de D a cada uno de los elementos de A. De este modo ya tenemos calculadas las nuevas coordenadas de los vértices. Recordemos que este programa lo hemos diseñado para que desplace la figura únicamente en el eje X. En consecuencia, el conjunto B que contiene las coordenadas Y no se ve afectado. Seguidamente se pasa control otra vez a la subrutina 500 donde se dibuja de nuevo la figura. Aunque la subrutina es la misma, al haber cambiado las coordenadas, el dibujo aparecerá desplazado respecto al original.

Observando el listado de la subrutina 600 veremos que es prácticamente idéntico al de la subrutina 500. Esto es lógico si recordamos que borrar líneas de la pantalla equivale a trazarlas en modo inverso. Las dos únicas variantes serán pues las instrucciones gráficas convencionales PUNTO y LINEA que se utilizan en modo inverso.

En este programa existen tres instrucciones DATA que contienen un total de 26 valores correspondientes a los vértices de una figura en forma de cruz. Puesto que se trata de una figura cerrada, las coordenadas del último punto coinciden con las del primero.

Ejecutando el programa nos aparecerá la figura en pantalla y a continuación nos pedirá el valor. Le contestaremos un número, por ejemplo 50. Veremos que desaparece la figura anterior y se dibuja de nuevo desplazada 50 posiciones hacia la derecha. Podemos repetir la ejecución y efectuar otros desplazamientos. Sin embargo, debemos tener cuidado de que el valor de las coordenadas no supere el tamaño de la pantalla, pues de lo contrario se producirá un error, a menos que nuestro ordenador esté dotado de un mecanismo automático de recortado de figuras, cosa poco frecuente en ordenadores pequeños.

Este programa nos ha permitido comprender cuál es el mecanismo básico de la traslación. Por tanto, ya podemos ampliarlo de modo que sea posible desplazar la figura en cualquier dirección. Esta transformación es muy sencilla en realidad. Añadiremos otra instrucción INPUT para obtener el desplazamiento en dirección vertical (eje Y), como puede ver en la línea 70. Por otra parte, en el bucle de la línea 90 le añadiremos el incremento de los elementos del conjunto B. Además, dotaremos al programa de la posibilidad de efectuar repetidas traslaciones, sin necesidad de empezar la ejecución de nuevo. Para ello modificamos la línea 110. Añadiendo también la línea 75. A continuación damos el listado del programa principal. Las subrutinas no varían. Recordemos que ésta es una de las ventajas de utilizarlas. Sus instrucciones permanecen invariables y se pueden utilizar en programas distintos.

Programa de «Traslación completa»

```
 10 REM Traslacion completa
 20 LET N=13: DIM A(N),B(N)
 30 FOR I=1 TO N
 40   READ A(I),B(I)
 50   NEXT I
 60 GOSUB 500
 70 INPUT "DESPL. X:";X1 : INPUT "DESPL. Y:";Y1
 75 IF X1=0 AND Y1=0 THEN STOP
 80 GOSUB 600
 90 FOR I=1 TO N:LET A(I)=A(I)+X1:
    LET B(I)=B(I)+Y1: NEXT I
100 GOSUB 500
110 GOTO 70
```

Entrada de los valores del desplazamiento

En la línea 70 existen dos instrucciones INPUT, una para el desplazamiento horizontal y la otra para el vertical. Según lo que respondamos en estas dos preguntas tenemos tres posibilidades. Para lograr una traslación en sentido horizontal, el valor de Y1 deberá ser cero. Para la traslación vertical, X1 valdrá cero. Para desplazamientos combinados, por ejemplo en diagonal, los dos valores X1 y Y1 tendrán un valor distinto de cero.

Puesto que si ambas variables valen cero no se produce traslación alguna, consideraremos este caso particular como una indicación de que el programa debe finalizar. Esta operación es la que realiza el IF de la línea 75. Notemos que empleamos el operador booleano AND para asegurar que las dos variables valen cero simultáneamente. En el bucle de la línea 90 se incrementan los valores de las coordenadas X e Y de todos los puntos. Lógicamente, si uno de los incrementos (X1 ó Y1) vale cero, la coordenada asociada no sufrirá variación. Finalmente, se ha añadido el GOTO de la línea 110 para que se puedan efectuar varias traslaciones en una sola ejecución del programa.

Al teclear el comando RUN, el programa dibuja la figura y nos pide el valor del desplazamiento horizontal. Le contestaremos un 50 por ejemplo, como en el caso anterior. A continuación nos pide el valor del desplazamiento vertical. Le contestaremos un 30. El programa dibuja entonces la figura trasladada hacia la derecha y hacia arriba. La ejecución prosigue y nos pregunta de nuevo el valor de X1. Si queremos trasladar el dibujo hacia la izquierda, le entraremos un valor negativo. Lo mismo se aplica si queremos realizar una traslación hacia abajo. Supongamos que contestamos ahora un −60 para X1 y un −40 para Y1. La figura quedará situada en el lado izquierdo inferior de la pantalla.

Conviene hacer notar que una traslación combinada se puede obtener también efectuando dos traslaciones simples. Por ejemplo, la traslación 50 y 30 equivale a una traslación 50 y 0 más otra traslación 0 y 30.

Una vez realizada una traslación, el programa nos pide otra vez nuevos valores para los desplazamientos. Contestándole los valores adecuados,

moveremos la figura por todo lo largo y ancho de la pantalla, teniendo siempre presente el no sobrepasar sus límites. Para detener el programa, responderemos dos ceros para los valores de ambos desplazamientos.



15.1.3 Escalado

Cambio de escala

La operación de cambio de escala nos permitirá aumentar o reducir el tamaño del dibujo representado en la pantalla. Como sabemos, la traslación se basaba en sumar un desplazamiento a cada uno de los vértices de que consta la figura. Para realizar el escalado, multiplicaremos las coordenadas por el factor de escala. También en este caso necesitaremos definir dos factores de escala, uno para el eje X y otro para el eje Y. Tomemos como ejemplo el triángulo de la figura 1 y supongamos que se desea doblar su tamaño. Entonces el factor de escala valdrá 2 (para ambos ejes). Las nuevas coordenadas pasarán a ser (2,2), (100,2) y (160,100). A la vista de estos valores, se comprueba fácilmente que este nuevo triángulo tiene el doble de tamaño que el inicial. Por otra parte, este ejemplo nos pone de manifiesto un problema que surge al efectuar el escalado. Al aumentar (o reducir) la figura, también aumenta (o disminuye) la distancia que separa a ésta del origen de coordenadas (punto (0,0) de la pantalla). El extremo izquierdo del triángulo, es decir, el vértice (1,1), está ahora situado en (2,2).

Ampliación de la figura respetando al máximo los límites de la pantalla

Hay que encontrar una solución a este problema, pues de lo contrario, si aplicamos el escalado a figuras muy alejadas del origen de coordenadas no nos será posible representarlas. Si tenemos un vértice situado en (150, 150) y queremos doblar el tamaño del dibujo, el resultado será (300,300) que sale fuera de los límites de la pantalla. Este fenómeno es independiente del tamaño inicial de la figura. Aunque ésta sea muy pequeña, si está muy alejada del origen, no se podrá aumentar su tamaño.

La solución a este problema consiste en suponer que la figura está lo más cerca posible del punto (0,0). ¿Cómo se puede realizar esto en la práctica? El procedimiento consiste en buscar el valor más pequeño de las coordenadas. Entonces se resta este valor de todos los puntos. Aquel punto cuya coordenada sea la mínima, valdrá ahora cero. Entonces se aplica el factor de escala. La coordenada mínima no se ve afectada, ya que cero multiplicado por cualquier número da siempre cero. Finalmente, se suma de nuevo esta coordenada mínima a todos los puntos, con lo cual la figura queda aumentada de tamaño pero situada en la misma posición que la original.

Con el triángulo de la figura 1, los pasos a seguir serían los siguientes. Supongamos que queremos doblar su anchura. En primer lugar se busca cuál es la coordenada X más pequeña. La coordenada mínima vale 1. Entonces se resta este valor de todas las demás. Las coordenadas resultantes son (0,1), (49,1) y (79,50). Al multiplicar por dos, las nuevas coordenadas son (0,1), (98,1) y (158,50). Sumando de nuevo la coordenada mínima ob-

tenemos (1,1), (99,1) y (159,50). De esta forma hemos doblado la anchura del dibujo sin modificar su posición en la pantalla. Si además de modificar la anchura, quisiéramos modificar la altura, entonces aplicaríamos el procedimiento anterior también a la coordenada Y. En suma, el proceso consiste en desplazar la figura hacia el origen, aplicar el factor de escala y desplazar otra vez la figura a su posición original.

El listado del programa es el siguiente:

Programa de «Escalado de figuras»

```
10 REM Escalado figuras
20 LET N=13: DIM A(N),B(N)
30 FOR I=1 TO N
40   READ A(I),B(I)
50   NEXT I
60 GOSUB 500
70 INPUT "ESCALA X:";EX
80 INPUT "ESCALA Y:";EY
90 IF EX=0 OR EY=0 THEN STOP
100 LET M1=A(1):LET M2=B(1)
110 FOR I=2 TO N
120   IF M1>A(I) THEN LET M1=A(I)
130   IF M2>B(I) THEN LET M2=B(I)
140   NEXT I
150 GOSUB 600
160 FOR I=1 TO N
170   LET A(I)=(A(I)-M1)*EX+M1
180   LET B(I)=(B(I)-M2)*EY+M2
190   NEXT I
200 GOTO 60
210 DATA 1,5, 3,5, 3,7, 5,7
220 DATA 5,5, 7,5, 7,3, 5,3
230 DATA 5,1, 3,1, 3,3, 1,3, 1,5
```

Las subrutinas 500 y 600 son las mismas que hemos utilizado en el programa anterior. La primera parte del programa, hasta la línea 60, ya nos es conocida. A continuación vienen dos instrucciones INPUT que nos piden los factores de escala para los ejes X e Y (líneas 70 y 80). Notemos que los factores de escala nunca pueden valer cero, ya que entonces todas las coordenadas también quedarían a cero. Por tanto, si uno de estos factores es cero, lo utilizaremos como condición de final de ejecución. El operador booleano empleado en este caso es el OR (línea 90).

Seguidamente se procede a determinar el valor mínimo de las coordenadas. Aprovechamos el mismo bucle para determinar ambos mínimos a la vez. Una vez determinados estos valores (M1 y M2) se borra el dibujo (subrutina 600) y se recalculan las nuevas coordenadas. Como podemos apreciar en las líneas 170 y 180, a cada coordenada se le resta el mínimo, se multiplica por el factor de escala y se le suma de nuevo el valor mínimo.

Finalmente se pasa control de nuevo a la línea 70 para que apliquemos nuevos factores de escala.

Si efectuamos un RUN, en pantalla aparecerá la figura en forma de cruz con un tamaño muy pequeño. El programa nos preguntará los valores del escalado. Contestaremos 2 y 2. La figura quedará ampliada dos veces. Si a la nueva pregunta contestamos 5 y 1, la figura queda muy ampliada horizontalmente, pero permanece con la altura invariable. Por tanto, la cruz ha quedado deformada. Restableceremos las proporciones iniciales escribiendo 1 y 5 como nuevos factores de escala. Si respondemos ahora 0.5 y 0.5, la cruz queda reducida a mitad de tamaño. Finalmente, contestando un cero para alguno de los factores, el programa se detiene.

En resumen, factores de escala mayores de 1 aumentan el tamaño. Si el factor vale 1, la figura permanece invariable y si el factor de escala es menor de 1, entonces se reduce el tamaño.

Efecto «zoom»

Una de las aplicaciones de la operación de escalado es la obtención del efecto «zoom». La palabra inglesa «zoom» proviene del campo de la cinematografía y designa la operación de ampliar (o reducir) de forma continua y rápida la imagen. Los ordenadores especializados en gráficos suelen incorporar la operación de zoom dentro del repertorio de funciones gráficas. Ahora bien, si nuestro ordenador no posee esta función, no significa que no podamos utilizarla. A continuación describiremos los pasos a seguir para elaborar un programa que realice un zoom.

Para obtener la sensación de que la figura va creciendo en la pantalla, es necesario borrar la figura anterior y dibujarla de nuevo ligeramente ampliada de forma muy rápida, de modo que lleguemos a engañar al ojo humano.

El obstáculo principal proviene de la velocidad de cálculo. Si ésta no es suficientemente elevada, se observarán discontinuidades entre cada ampliación. De hecho, los ordenadores especializados realizan estos cálculos en código máquina a fin de conseguir la máxima velocidad. Sin embargo, el BASIC, por su propia naturaleza es mucho más lento que el código máquina. Para superar esta limitación nuestro programa servirá únicamente para figuras con pocos vértices, de modo que el tiempo total de cálculo sea reducido.

Una cuestión importante en el efecto zoom es que la figura crezca hacia el exterior de un punto imaginario situado en el centro de la figura. No obstante, en este caso no disponemos de tiempo para buscar el mínimo, restarlo y calcular de nuevo las coordenadas. En su lugar haremos lo siguiente. Tomaremos las coordenadas suponiendo que el centro de la figura coincide con el origen (punto (0,0) de la pantalla). Esto dará lugar a coordenadas positivas y negativas. Todas las operaciones de cambio de escala se efectuarán sobre estos valores. En el momento de dibujar las líneas, y sólo entonces, se sumará a las coordenadas un desplazamiento fijo para que la figura quede centrada en la pantalla.

A continuación se da el listado del programa:

Programa de «Efecto zoom»

```
10 REM Efecto zoom
20 DIM A(4),B(4)
```

```
30 FOR I=1 TO 4
40   READ A(I),B(I)
50   NEXT I
60 FOR J=1 TO 20
70   GOSUB 200
80   FOR K=1 TO 10:NEXT K
90   GOSUB 300
100  FOR I=1 TO 4
110    LET A(I)=A(I)*1.2:LET B(I)=B(I)*1.2
120    NEXT I
130  NEXT J
140 STOP
150 DATA 0,2, 2,-1, -2,-1, 0,2
200 LET X=A(1)+100:LET Y=B(1)+50
210 PUNTO(X,Y)
220 FOR I=2 TO 4
230   LET X=A(I)+100:LET Y=B(I)+50
240   LINEA(X,Y)
250   NEXT I
260 RETURN
300 LET X=A(1)+100:LET Y=B(1)+50
310 PUNTO(X,Y),I
320 FOR I=2 TO 4
330   LET X=A(I)+100:LET Y=B(I)+50
340   LINEA(X,Y),I
350   NEXT I
360 RETURN
```

Se ha escogido la figura más sencilla, un triángulo, de modo que los cálculos se efectúen con la máxima rapidez. Las subrutinas 200 (trazado) y 300 (borrado) son muy parecidas a las que ya conocemos, con la excepción de que suman un desplazamiento a las coordenadas. Al ejecutar el programa veremos que aparece un pequeño triángulo en el centro de la pantalla que va creciendo progresivamente. La velocidad de crecimiento está controlada por el bucle de espera de la línea 80. Cambiando su valor, modificaremos la velocidad del efecto zoom.

15.1.4 Rotación

Rotación de figuras

Esta es la última operación bidimensional que nos queda por ver. Con ella cambiaremos la orientación de un dibujo respecto a la pantalla. Para efectuar los cálculos emplearemos las funciones trigonométricas SIN y COS. Hay un problema parecido al que existía en la operación de escalado. Si se aplican directamente las funciones seno y coseno, la rotación se produce alrededor del punto (0,0) de la pantalla y en cambio nosotros queremos que gire sobre sí misma. La solución consistirá en tomar las coordenadas de la figura como si estuviera centrada en el origen de coordenadas

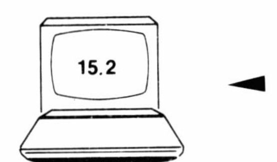
y luego añadirle el desplazamiento para situarla en el lugar correcto de la pantalla.

El listado es el siguiente:

Programa de «Rotación»

```
 10 REM Rotacion
 20 LET N=4: DIM A(N),B(N)
 30 FOR I=1 TO N
 40   READ A(I),B(I)
 50   NEXT I
 60 GOSUB 200
 70 INPUT "ANGULO:";R
 80 IF R=0 THEN STOP
 90 LET R=PI*R/180
100 LET C=SIN(R): LET D=COS(R)
110 GOSUB 300
120 FOR I=1 TO N
130   LET T=A(I)*D+B(I)*C
140   LET B(I)=-A(I)*C+B(I)*D: LET A(I)=T
150   NEXT I
160 GOTO 60
170 DATA 0,20, 20,-10, -20,-10, 0,20
```

Se aprovechan las subrutinas 200 y 300 del programa anterior. A la pregunta de la línea 70 responderemos un ángulo de giro que sea mayor de cero y menor de 360 grados. En la línea 90 se transforma el valor del ángulo de grados a radianes. Finalmente las líneas 130 y 140 calculan las nuevas coordenadas giradas respecto al punto central. La rotación se efectúa en el mismo sentido que el movimiento de las agujas del reloj.



15.2 DIBUJOS TRIDIMENSIONALES

Todos los conceptos que hemos visto para los espacios de dos dimensiones pueden ampliarse a espacios de tres dimensiones.

Para representar un punto en los ejes de coordenadas bidimensionales necesitamos dos valores que corresponden a la distancia del punto a cada uno de los ejes. Para un espacio de tres dimensiones definiremos unos ejes cartesianos también tridimensionales como los representados en la figura 2, en donde existe un tercer eje Z además de los dos habituales X e Y. Las coordenadas de un punto en el espacio se representan con tres valores en lugar de dos como hasta ahora.

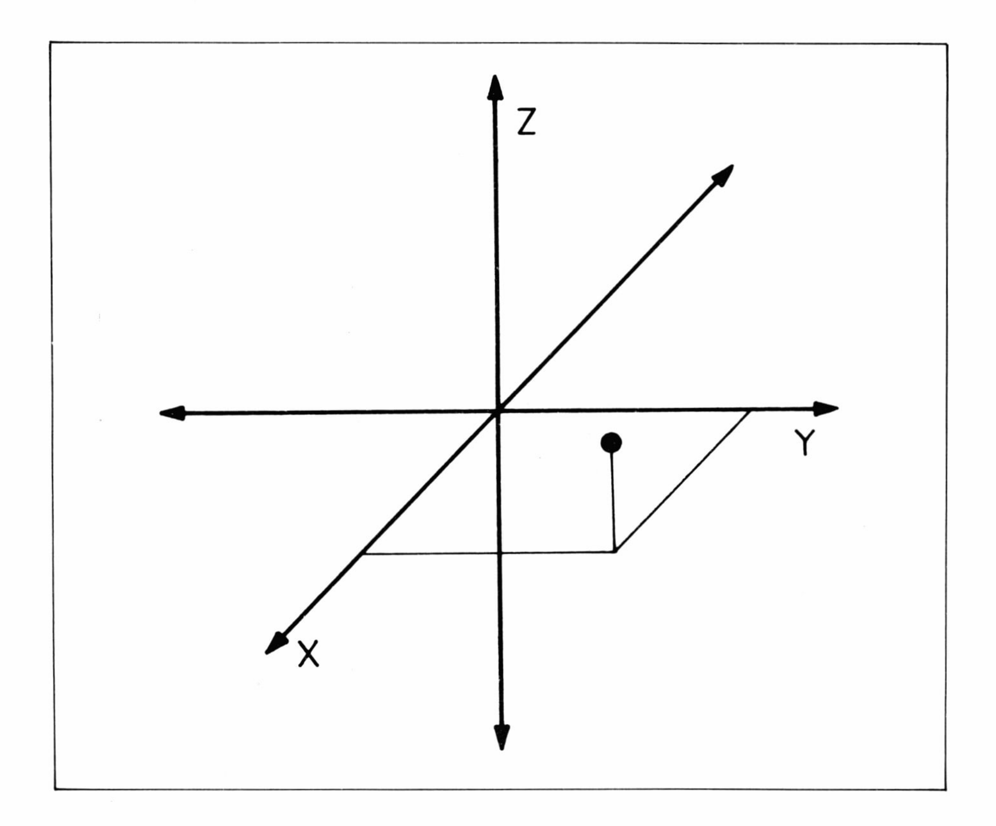


Figura 2. Ejes de coordenadas tridimensionales y coordenadas de un punto situado en el espacio.

15.2.1 Perspectiva

Por el momento ya sabemos determinar las coordenadas de un cuerpo en el espacio. El problema surge cuando se intenta representar este cuerpo sobre la superficie bidimensional de la pantalla.

¿Cómo pasar de tres coordenadas a dos? De hecho este mismo problema se presenta en el mundo del dibujo y de la pintura, puesto que un cuadro también es una representación bidimensional.

La perspectiva en la representación tridimensional

La solución consiste en utilizar alguno de los métodos de perspectiva (caballera, cónica, axonométrica, etc.). Aunque existen una serie de conceptos y fórmulas matemáticas para la realización de perspectivas, su estudio escapa de los objetivos de este curso. En nuestros programas utilizaremos la perspectiva únicamente de forma intuitiva.

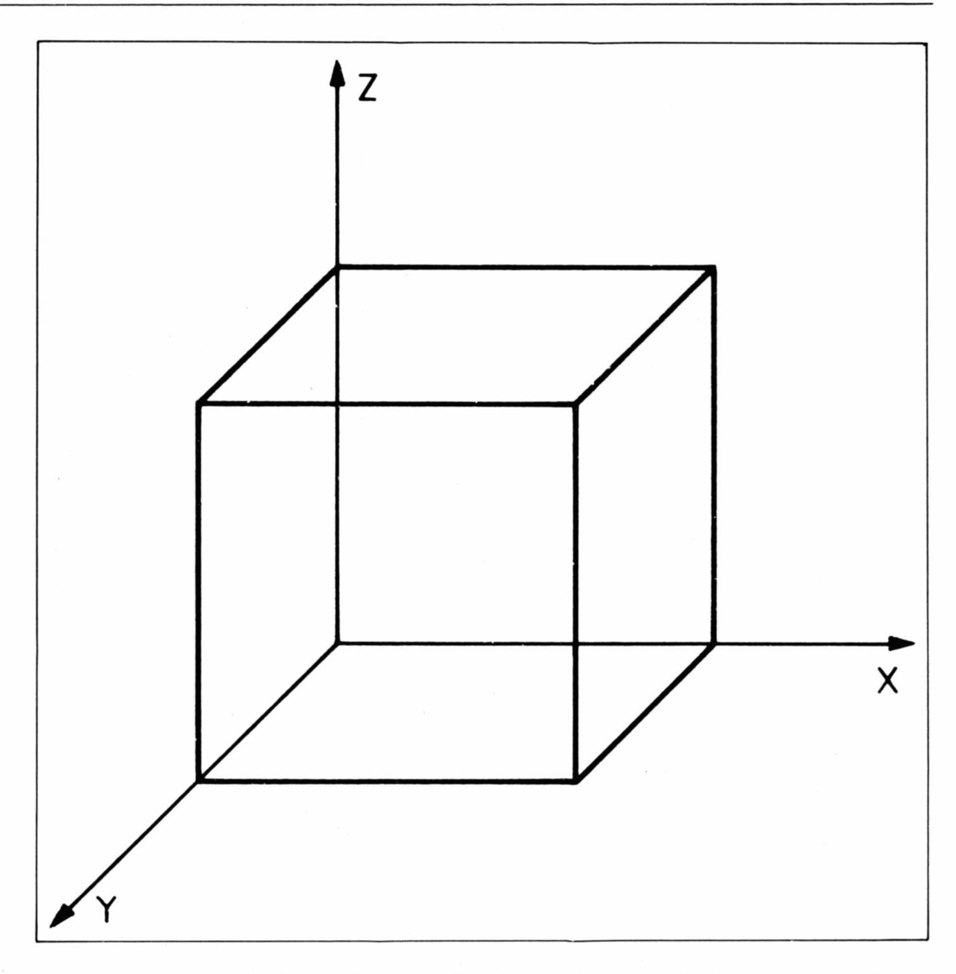
En resumen, la perspectiva es una técnica de representación de objetos sobre una superficie de forma que aparezcan tal como se verían si los mirásemos con un ojo desde una distancia determinada. Un ejemplo de perspectiva lo tenemos en la figura 3.

15.2.2 Representación de superficies

Método de la retícula

Un problema asociado a la perspectiva es la representación de superficies. En los objetos muy simples como por ejemplo un cubo (Fig. 3), el propio contorno del objeto es suficiente para representar las superficies. Sin

Figura 3. Vista en perspectiva de un cubo.

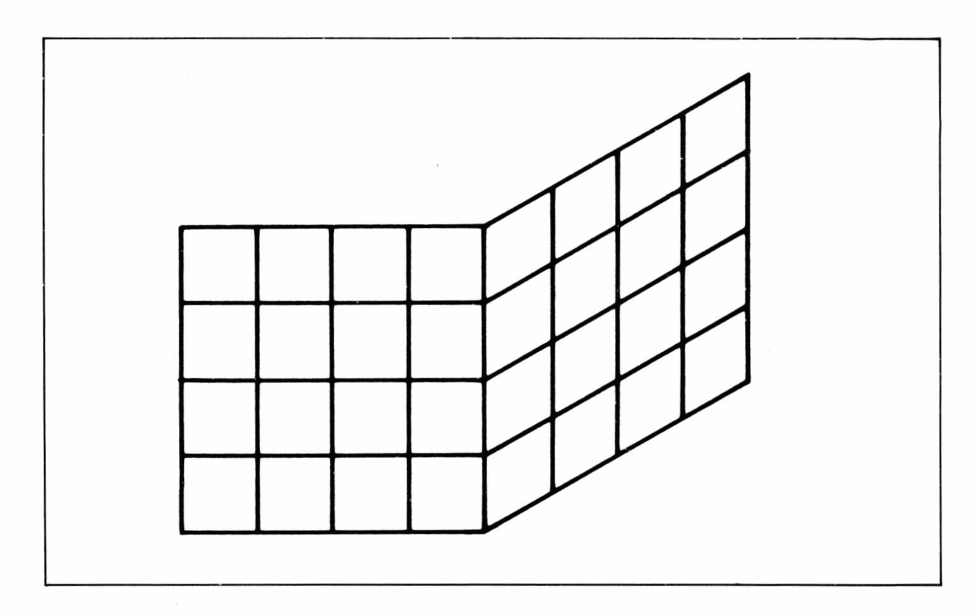


embargo, para objetos más complicados este procedimiento resulta insuficiente. Entonces se suele recurrir a la técnica del sombreado, de modo que diferentes tonos de gris distingan a una superficie de otra. Otra técnica consiste en utilizar un conjunto de líneas entrecruzadas de manera que formen una retícula o enrejado. Para simplificar, podríamos decir que este método se basa en suponer que las caras o superficies de un objeto están formadas por tela metálica. De esta manera, cuando aparecen dobleces en la retícula es que no se trata de una arista (separación entre caras). En la figura 4 se muestra un ejemplo del procedimiento de la retícula para representar dos planos que se cruzan en el espacio. Este método tiene la ventaja de que es sumamente fácil de programar en un ordenador; por esta razón es muy utilizado en el diseño asistido por ordenador. A continuación vamos a dibujar el cubo por este sistema.

Para que vea con más claridad el proceso, iremos construyendo el programa parte por parte. Después de cada parte usted deberá ejecutar el programa y así poder observar en pantalla el resultado de lo que va haciendo. En el texto le incluimos la figura de lo que usted está realizando, destacando con línea más gruesa la parte que en ese momento está programando.

Comenzaremos por dibujar las líneas horizontales, incluyendo su continuación inclinada (Fig. 5). Es lo que vamos a hacer en las líneas 20 a 50.

Figura 4. Utilización de una retícula para representación de superficies tridimensionales.



En la figura hemos puesto también las coordenadas para que pueda identificar el origen y el final de las rectas que dibuja.

Programa de «Superficies Tridimensionales»

```
10 REM Superficies tridimensionales
20 FOR Y=10 TO 70 STEP 10
30   PUNTO(10,Y)
40   LINEA(50,Y): LINEA(90,Y+20)
50   NEXT Y
```

Ejecute ahora esta parte del programa y verá que le traza las rectas de la figura 5.

A continuación trazamos las rectas verticales correspondientes al plano frontal (Fig. 6). Es lo que hacemos en las líneas 60 a 90.

```
60 FOR X=10 TO 50 STEP 10
70   PUNTO(X,10)
80   LINEA(X,50)
90   NEXT X
```

Ejecute de nuevo el programa y verá en pantalla las rectas de la figura 6.

A continuación trazamos las rectas verticales del segundo plano en las líneas 100 a 130 (Fig. 7).

```
100 FOR X=60 TO 90 STEP 10
110   PUNTO(X,X/2-15)
120   LINEA(X,X/2+25)
130   NEXT X
```

Figura 5. Trazado de las rectas horizontales.

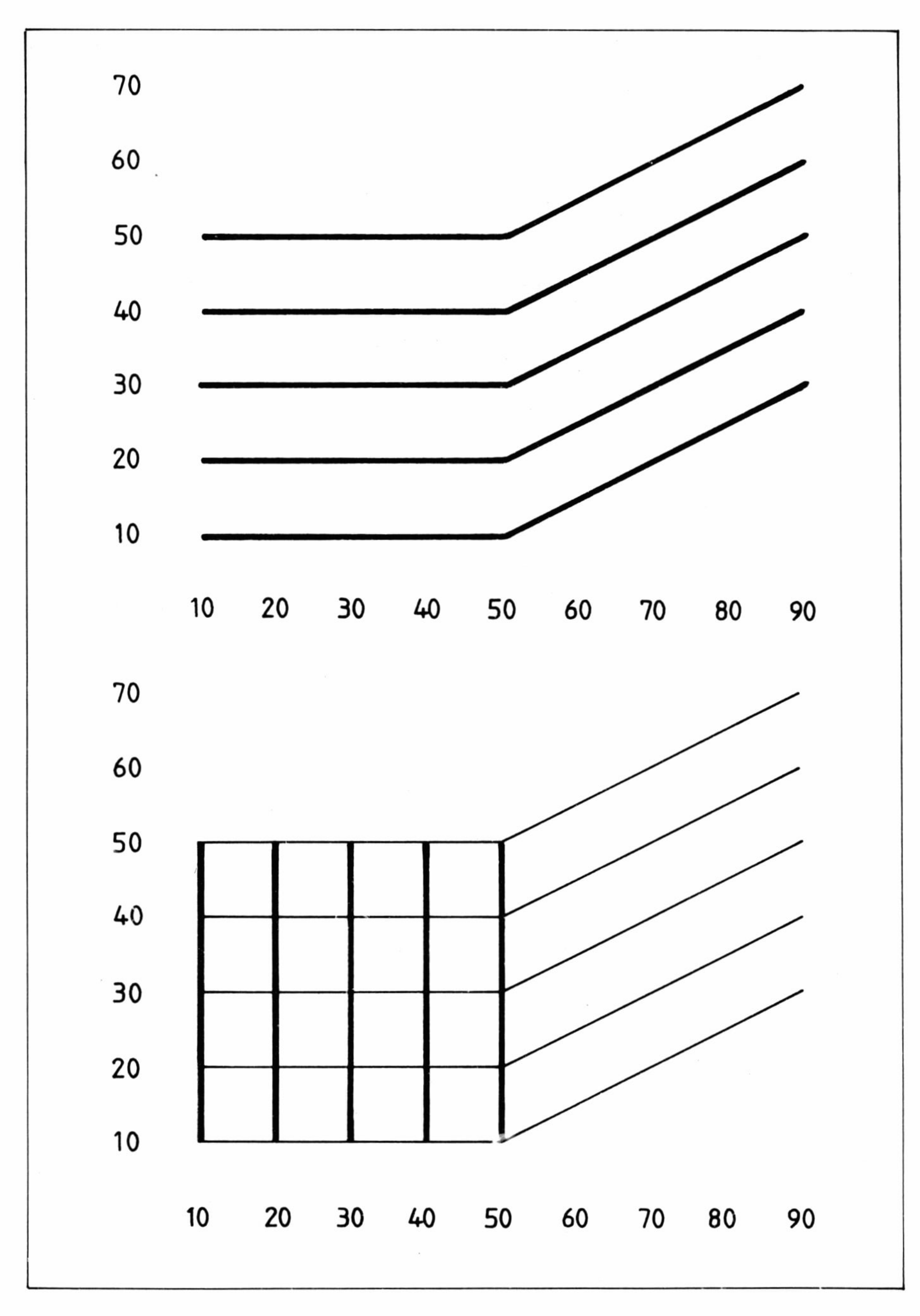


Figura 6. Trazado de las rectas verticales del plano frontal.

El valor aparece para *Y* (X/2−15) es un artilugio para que el valor de *Y* vaya aumentando de cinco en cinco, cuando el de *X* aumenta de diez en diez. Así vea que cuando *X* vale 60 en la primera vuelta del bucle, el valor de *Y* es: 60/2−15=15. En la segunda vuelta del bucle en que *X* vale 70, el valor de *Y* sería: 70/2−15=20, que es la altura donde comienza ahora la línea que trazamos. Puede verlo en la figura 7. Estos pequeños trucos se los encontrará en varias líneas del programa que estamos escribiendo.

Ahora vamos a dibujar la superficie superior del cubo. Comenzamos trazando las rectas que puede ver en la figura 8.

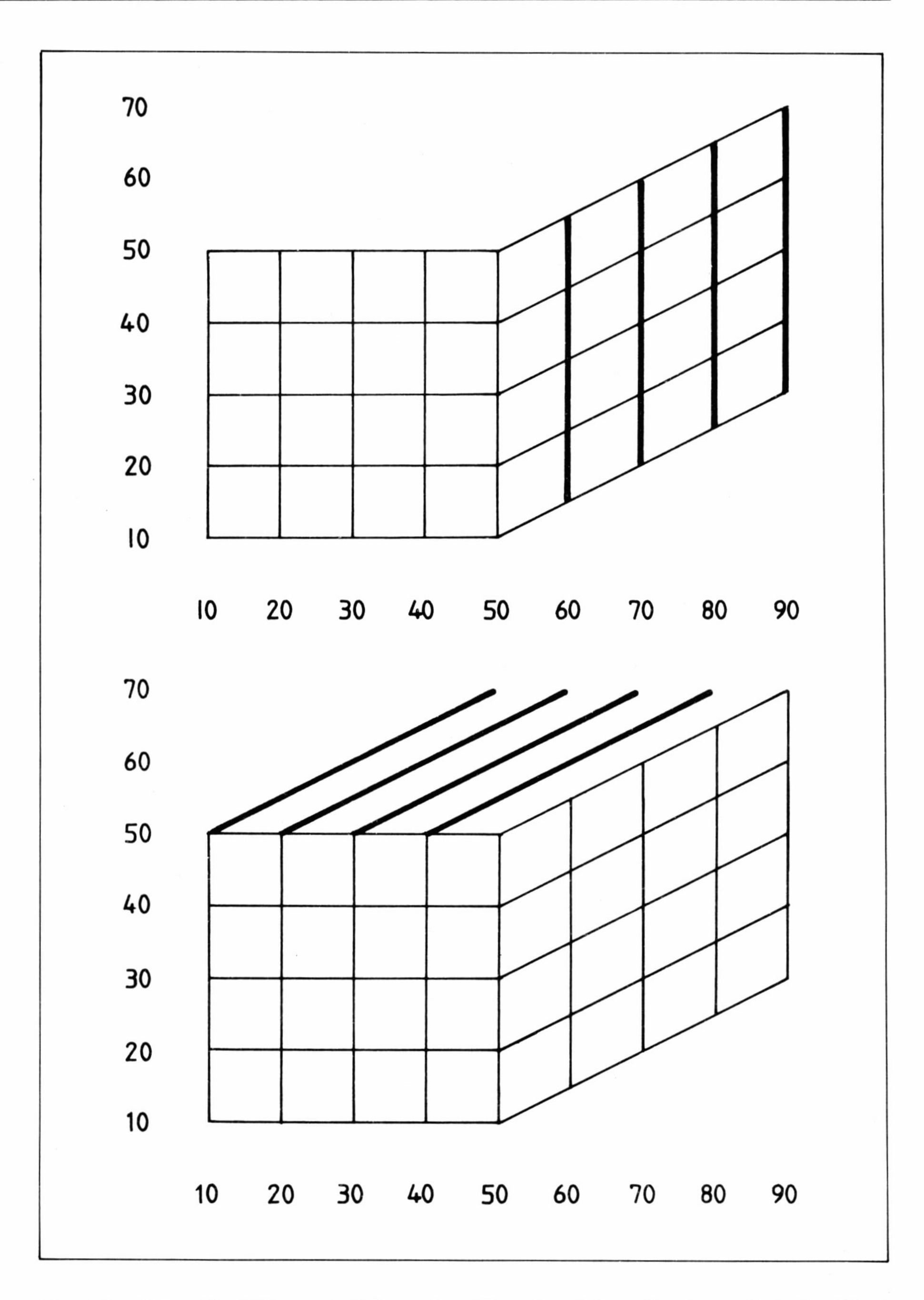


Figura 7. Trazado de las rectas verticales del segundo plano.

Figura 8. Trazado de las rectas verticales del plano superior.

```
140 FOR X=10 TO 40 STEP 10
150    PUNTO(X,50)
160    LINEA(X+40,70)
170    NEXT X
```

Por último trazaremos las rectas horizontales correspondientes a la cara superior del cubo en las líneas 180 a 210 (Fig. 9).

```
180 FOR Y=55 TO 70 STEP 5
190   PUNTO(2*Y-90,Y)
200   LINEA(2*Y-50,Y)
210   NEXT Y
```

Este programa que acabamos de escribir nos representa en pantalla las tres superficies exteriores visibles de un cubo visto en perspectiva.

En realidad, la perspectiva empleada no es totalmente correcta, pero se ha tomado esta decisión a fin de simplificar el programa.

Al dibujar el cubo no se han mostrado las caras que están situadas en la parte posterior. De esta forma, el cubo tiene una apariencia sólida. Por el contrario, si se mostrasen todas las caras, entonces tendría el aspecto de un objeto transparente.

Ahora dibujaremos estas caras ocultas siguiendo el mismo proceso. En las figuras 10 a 14 tenemos las rectas que iremos trazando para representar estas caras ocultas.

A continuación le presentamos el programa para hacerlo. Sin embargo, antes de copiar el programa que le ofrecemos, intente usted escribirlo, siguiendo las pautas de las rectas a dibujar, que tiene en las figuras 10 a 14. Vaya por partes y ejecute el programa después de escribir cada parte.

Comenzaremos también por dibujar las rectas horizontales de las caras ocultas (Fig. 10) y que, como en el resto de las figuras aparecen con trazo grueso. Hemos comenzado por la línea 300 a fin de separar las dos partes del programa.

```
300 FOR Y = 10 TO 40 STEP 10
310 PUNTO (10,X)
320 LINEA (50,Y+20) : LINEA (90,Y+20)
330 NEXT Y
```

Después trazaremos las rectas de la cara posterior (Fig. 11) en las líneas 340 a 370.

```
340 FOR X = 50 TO 80 STEP 10
350 PUNTO ( X,30)
360 LINEA (X,70)
370 NEXT X
```

A continuación trazamos las rectas verticales de la cara lateral izquierda (Fig. 12).

```
380 FOR X= 20 TO 40 STEP 10
390 PUNTO ( X,X/2+45)
400 LINEA ( X,X/2+45)
410 NEXT X
```

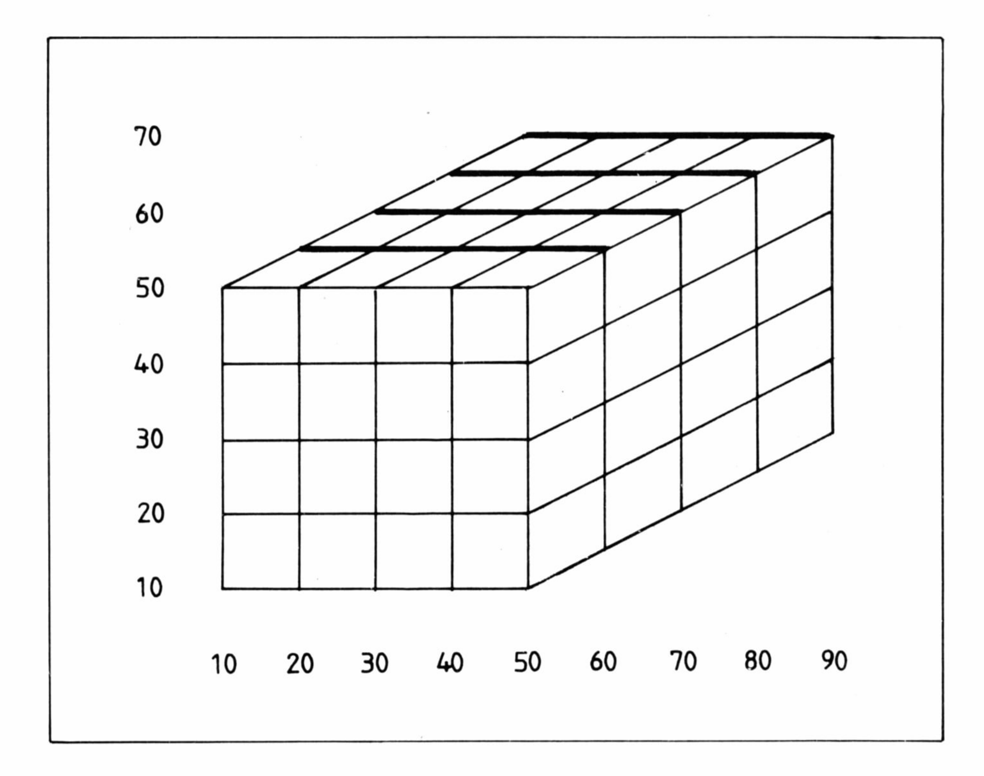
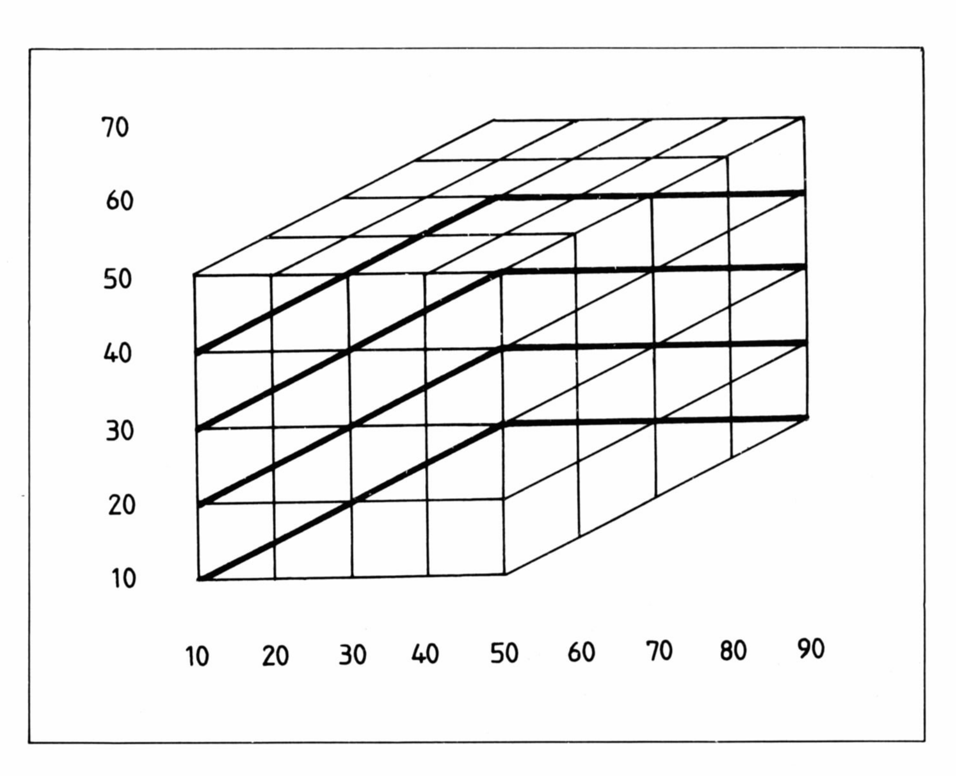


Figura 9. Trazado de las rectas horizontales del plano superior.

Figura 10. Trazado de rectas horizontales de la cara oculta.



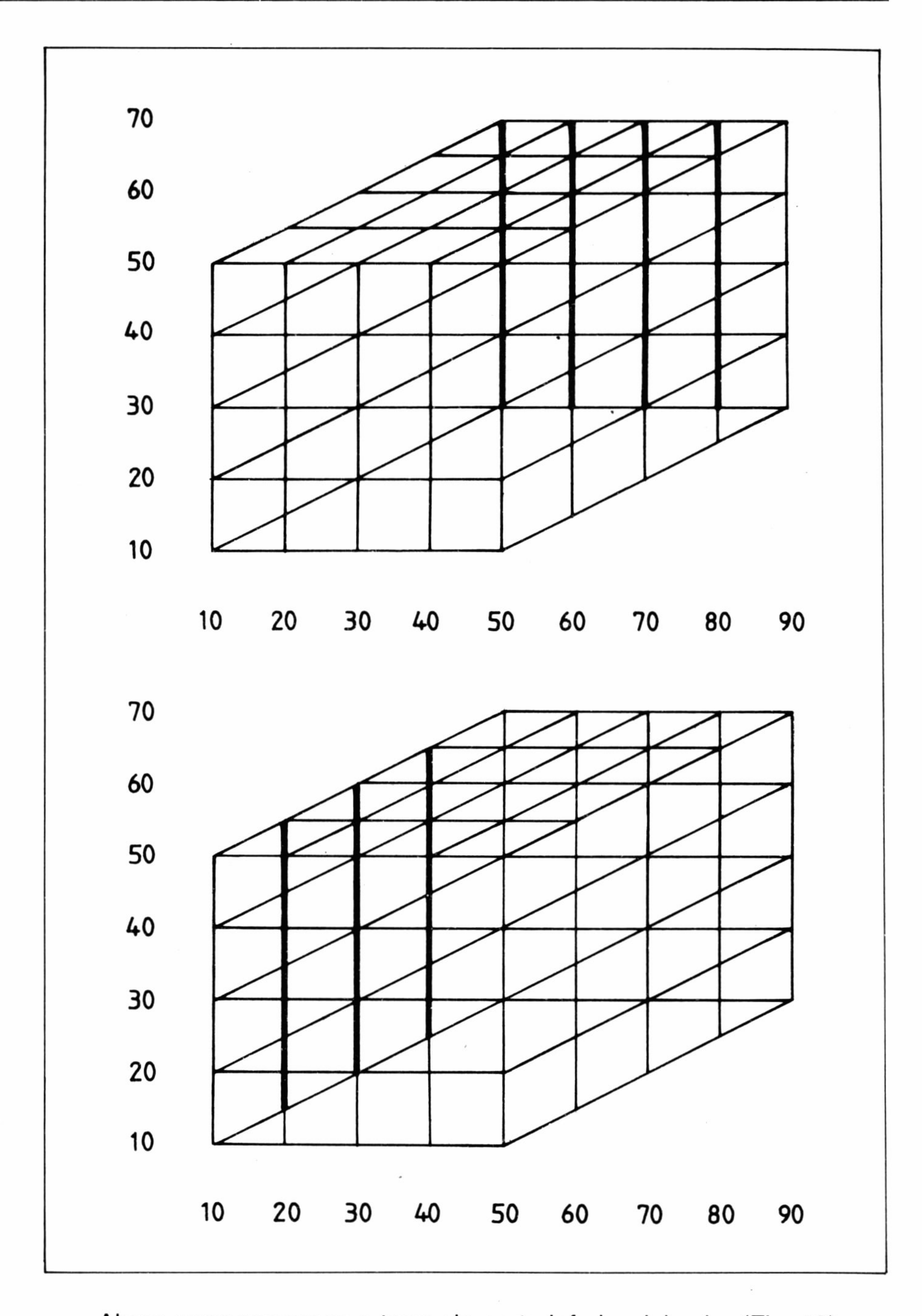


Figura 11. Trazado de las rectas verticales de la cara posterior.

Figura 12. Trazado de las rectas verticales de la cara posterir izquierda.

Ahora comenzaremos a trazar la parte inferior del cubo (Fig. 13).

```
420 FOR X=20 TO 40 STEP 10
430 PUNTO (x,10)
440 LINEA (X+40,30)
450 NEXT X
```

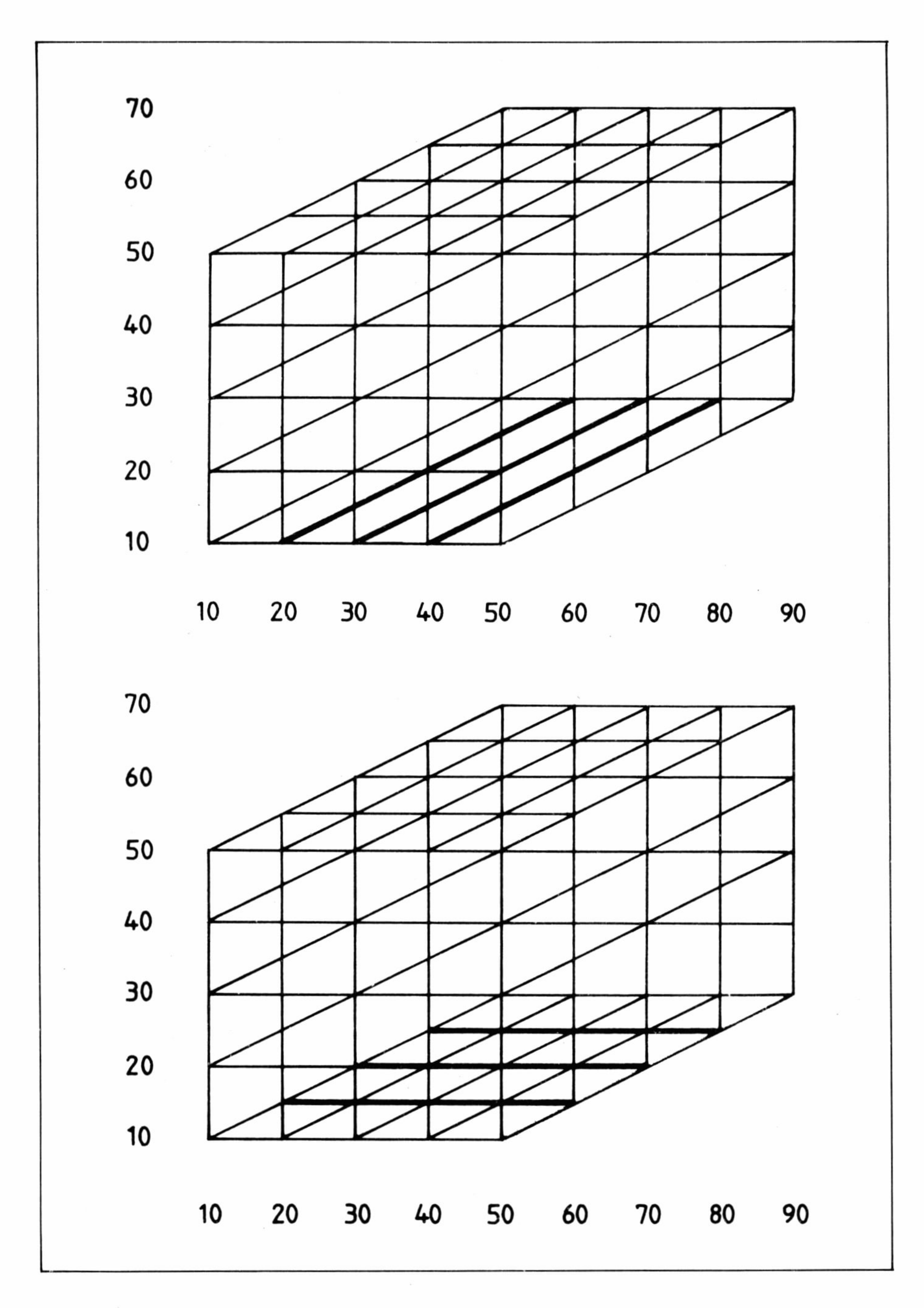


Figura 13. Rectas verticales de la parte inferior del cubo.

Figura 14. Rectas horizontales de la parte inferior del cubo.

Ahora ya sólo nos falta trazar las líneas horizontales de la superficie inferior (Fig. 14).

```
460 FOR Y=15 TO 25 STEP 5
470 PUNTO(2*Y-10,Y)
480 LINEA(*Y+30,Y)
490 NEXT Y
```

15.2.3 Representación de funciones

De la misma forma que podemos realizar diagramas X-Y de funciones, también es posible realizar diagramas X-Y-Z, es decir, una representación tridimensional de una función. Naturalmente será necesario usar la técnica de la perspectiva para situar el diagrama en la pantalla. Para representar la superficie se utilizará la técnica del retículo. Aquí aparece de nuevo el problema de las líneas ocultas. Puesto que la superficie representada tendrá zonas elevadas y zonas deprimidas, será necesario que las líneas que pasen detrás de otras zonas no se dibujen a fin de que la superficie dé la sensación de solidez.

Figuras tridimensionales

Un programa que sea capaz de representar cualquier tipo de función es muy difícil de elaborar. Por ello construiremos uno simplificado. Este programa realizará diagramas de funciones X-Y que se girarán alrededor del eje Z, de modo que adquieran un aspecto tridimensional, del mismo modo que girando una circunferencia se obtiene una esfera y girando un triángulo se obtiene un cono. El listado es el siguiente:

Programa de «Dibujo tridimensional»

```
 10 REM Dibujo Tridimensional
 20 DEF FNA(Z)=32*EXP(1-0.005*Z*Z)
 30 INPUT "RESOLUCION (1-10):";R
 40 FOR X=-100 TO 100
 50 LET J=0
 60 LET X2=X*X: LET X3=X+110
 70 LET V=R*INT(SQR(1E4-X2)/R)
 80 FOR Y=V TO -V STEP -R
 90   LET Z=INT(80+FNA(SQR(X2+Y*Y))-0.707*Y)
100   IF Z<J THEN GOTO 130
110   LET J=Z
120   PUNTO(X3,Z)
130   NEXT Y
140 NEXT X
150 STOP
```

Este programa realiza un gráfico como el que se muestra en la figura 15, pero sin los ejes de coordenadas. Aunque el programa es bastante corto, es notablemente complicado puesto que hace uso simultáneamente de muchos conceptos matemáticos. A causa de que el número de operaciones que efectúa es muy elevado, la realización del gráfico emplea un tiempo de varios minutos. Es más, si quiere que realice un gráfico con resolución 1 puede aprovechar para tomarse un café con calma o hacer cualquier otra cosa, mientras el programa continúa dibujando, pues le dará tiempo. Dada la complejidad del programa, daremos solamente una explicación simplificada.

En la línea 20 se establece la función a graficar. Tal vez sea conveniente repasar los conocimientos sobre la instrucción DEF explicada en la

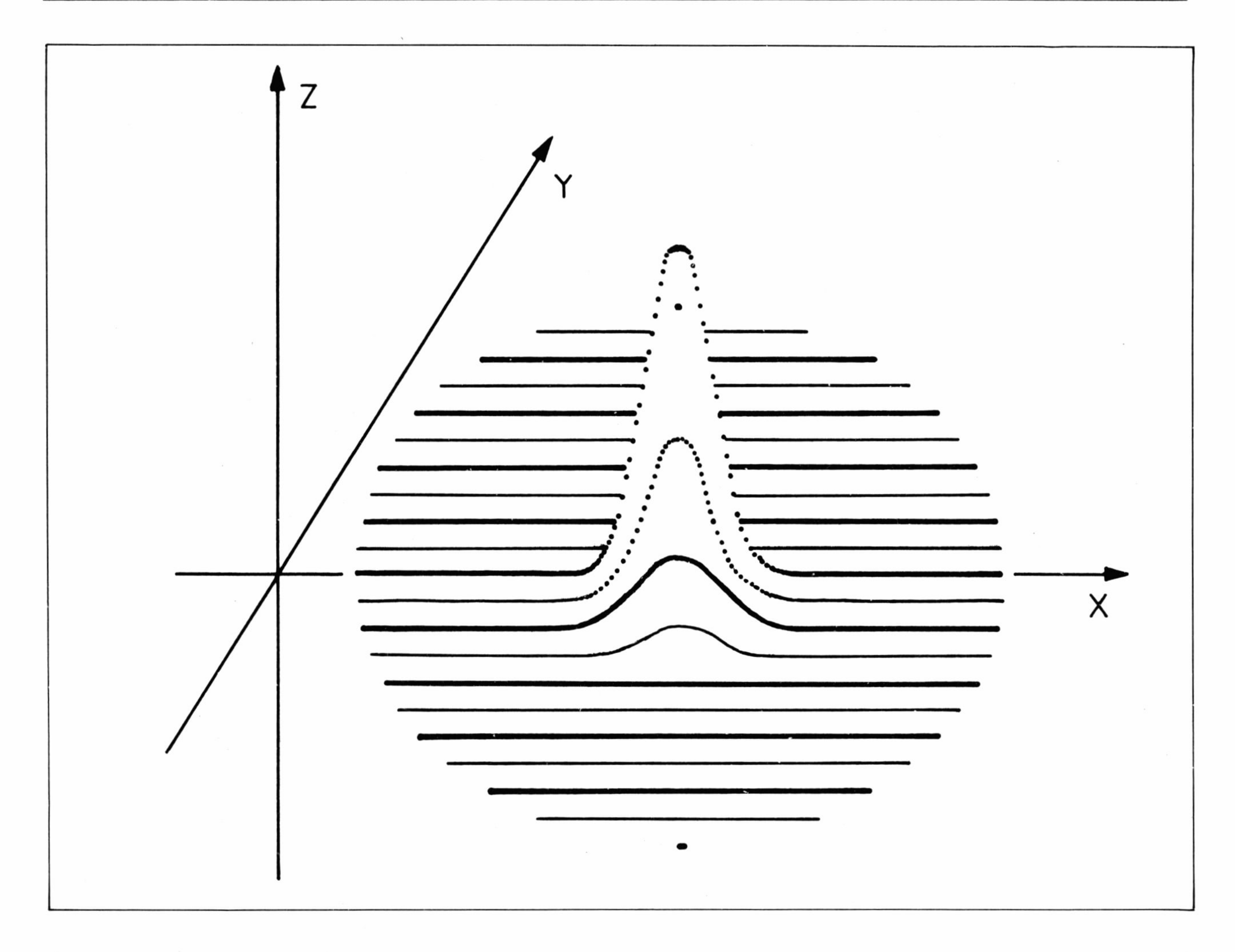


Figura 15. Perspectiva de una función tridimensional.

en el último capítulo del tomo anterior. Los aficionados a la estadística reconocerán enseguida que se trata de una función gaussiana (campana de Gauss). Si éste no es nuestro caso, consideremos simplemente que se trata de una función arbitraria como lo podía ser cualquier otra.

El gráfico se construye únicamete a base de puntos, sin utilizar líneas (luego explicaremos por qué). Estos puntos estarán muy juntos o muy separados según la calidad que queramos darle al gráfico. Este grado de proximidad lo determinaremos mediante la variable R en la línea 30. Notemos que en el texto de la instrucción INPUT aparece la palabra «resolución». Es perfectamente correcto utilizar esta palabra, puesto que, según la definición, indica el número de puntos empleados en hacer el gráfico, que es precisamente lo que realiza R. Hay que tener en cuenta que cuanto menor sea el valor de R, mayor será el tiempo de cálculo. Por tanto, para las primeras pruebas utilizaremos un valor de 10.

Este programa dibuja de izquierda a derecha, como se ve en el bucle de la línea 40 que va desde −100 hasta 100. Para cada posición de X dibuja una serie de puntos según el eje Y empezando desde abajo hacia arriba. Simultáneamente se calcula la coordenada Z y se comprueba si aquel punto

queda tapado por los puntos previos. Si no es así, se dibuja el punto. Se utilizan puntos y no rectas para construir el dibujo precisamente por esta razón. En una recta es más difícil evaluar qué trozo es visible y qué trozo queda escondido por el resto del dibujo.

Cambiando adecuadamente la función definida en la línea 20, representaremos otras funciones. Por ejemplo, cambiando la línea 20 por:

```
20 DEF FNA(Z)=32*SIN(Z/6)
```

obtendremos en pantalla una figura de forma ondulada, bastante espectacular. Utilizando la función:

```
20 DEF FNA(Z)=32*SQR(Z*0.005)
```

se obtiene una superficie con un pequeño hoyo en el centro.

Siguiendo este procedimiento podemos representar cualquier función, procurando siempre que no supere el tamaño de la pantalla ya que entonces se produciría un error.

RESUMEN

Para trabajar con figuras que no sean polígonos regulares, memorizaremos las coordenadas de los vértices. Si la figura tiene líneas curvas, se descompondrán éstas de una serie de pequeñas líneas rectas y se memorizarán sus vértices.

Las operaciones que se pueden aplicar sobre una figura son: la traslación, el escalado y la rotación. Todas estas operaciones se basan en el mismo procedimiento, que consiste en borrar la figura inicial, calcular las nuevas coordenadas de los vértices y dibujar la figura.

La operación de traslado es la más sencilla y se efectúa añadiendo un valor constante a todas las coordenadas. Se puede añadir simultáneamente un valor para las coordenadas X y otro para las coordenadas Y de modo que la figura se traslade al mismo tiempo sobre el eje horizontal y el vertical. Para realizar el escalado se multiplicarán las coordenadas por un valor fijo. Aquí también se puede aplicar un valor distinto para las coordeadas X y las Y. Para que la figura no vea alterada su posición original en la pantalla, se efectuará un traslado al origen de coordenadas, se aplicará el escalado y se trasladará de nuevo. Para calcular la rotación de una figura utilizaremos las funciones trigonométricas SIN y COS. Aquí también realizaremos un traslado respecto al origen de coordenadas.

Las figuras tridimensionales se representarán mediante la perspectiva. A su vez, las superficies se representarán mediante la técnica

del sombreado o utilizando una retícula. Si se dibujan todas las caras, las figuras adquieren un aspecto transparente. Por el contrario, si las caras ocultas no se representan, la figura da la sensación de solidez.

Utilizando la perspectiva se pueden realizar también diagramas tridimensionales de funciones X-Y-Z sobre una pantalla. Como que la función será una superficie en lugar de una línea como en los diagramas X-Y, entonces habrá que utilizar la técnica de la retícula para representarla.

EJERCICIOS DE AUTOCOMPROBACION

Completar las frases siguientes:

1. Memorizando las coordenadas de los .... conseguiremos dibujar cualquier tipo de figura.

2. Las operaciones más importantes que se pueden realizar sobre un dibujo bidimensional son la traslación, el .... y la rotación.

3. Los desplazamientos negativos en la operación de traslación originan un movimiento de la figura hacia la .... o hacia abajo.

4. Mediante el cambio de escala podemos ampliar o .... el tamaño de una figura.

5. Si se aplica un factor de escala distinto para cada eje, entonces la figura queda ....

6. Un factor de escala inferior a 1 produce una figura de tamaño .... al original.

7. La rotación cambia la .... de una figura respecto a la pantalla.

8. Angulos positivos originan una rotación en .... que las agujas del reloj.

9. La técnica del retículo se utiliza para representar .... de figuras tridimensionales.

10. Si en una figura tridimensional se representan todas sus caras, dicha figura dará la sensación de ....

11. Para representar funciones tridimensionales en la pantalla emplearemos también la técnica de la .... junto con el método de la retícula.

12. Una función tridimensional representada en unos ejes X-Y-Z tiene la forma de una .... en lugar de una línea como las bidimensionales.

13. Cuanto más junto estén los puntos de la superficie, ... será la calidad del gráfico.

14. Para saber si una zona de la superficie de una función queda oculta se comprueban los .... calculados previamente.

15. El número de cálculos a realizar para un dibujo tridimensional es mucho .... que para un dibujo bidimensional.

Encierre en un círculo la letra que corresponda a la alternativa correcta.

16. Al dibujar una figura de tipo irregular empezaremos

a) Colocando el punto o vértice inicial

b) Trazando una recta horizontal

c) Trazando una recta vertical

d) Colocando un punto en las coordenadas (0,0)

17. La operación de traslación de una figura se puede realizar

a) Solamente sobre las coordenadas X

b) Simultáneamente sobre las dos coordenadas

c) Solamente en sentido vertical

d) Solamente hacia la derecha

18. Para borrar una figura dejando sin variar el resto de la pantalla
    a) Se utiliza la instrucción CLS
    b) Se escriben espacios en blanco
    c) Se dibuja otra vez la figura utilizando el modo inverso
    d) Se dibuja otra vez la figura empezando por la última coordenada

19. Antes de aplicar el factor de escala, hay que
    a) Calcular la suma de los vértices
    b) Buscar el valor máximo
    c) Dividir el factor de escala por la coordenada máxima
    d) Trasladar la figura a la posición (0,0), restándole la coordenada mínima

20. Para que una figura efectúe una rotación sobre sí misma, sus coordenadas se tomarán suponiendo que
    a) El vértice izquierdo está en (0,0)
    b) El vértice derecho está en (0,0)
    c) El centro de la figura está en (0,0)
    d) Que el centro de la figura coincide con el centro de la pantalla

21. Utilizando el método de la retícula, la separación entre dos caras se representa
    a) Aumentando la separación de las líneas
    b) Doblegando las líneas de la retícula
    c) Disminuyendo el número de líneas de la retícula
    d) Cambiando el color de las líneas

22. Para representar un objeto sólido utilizamos la perspectiva y
    a) No dibujamos las caras ocultas
    b) Trazamos líneas gruesas para las caras posteriores
    c) La retícula será más tupida para las caras visibles
    d) Utilizaremos únicamente la retícula para las caras posteriores

23. Cuanto más juntas estén las líneas de una retícula
    - *a*) La representación de la superficie se hará con mayor precisión
    - *b*) La figura tendrá mayor sensación de transparencia
    - *c*) De mayor tamaño será la figura
    - *d*) Menos tiempo tardará el ordenador en realizar los cálculos

24. El método de la retícula es muy utilizado en ordenadores porque
    - *a*) Es fácil de programar
    - *b*) No tiene en cuenta la perspectiva
    - *c*) Sólo sirve para superficies verticales
    - *d*) No existe otro método

25. Para dibujar una función tridimensional es más fácil construir un programa que trace puntos en lugar de líneas porque
    - *a*) Es mucho más rápido de cálculo
    - *b*) El objeto es más perfecto
    - *c*) Así no hay que calcular la perspectiva
    - *d*) Sería difícil calcular la parte visible de cada línea trazada



15.3 GRAFICOS EN COLOR

Una posibilidad muy atractiva que suelen ofrecer los ordenadores modernos es la utilización de colores para construir gráficos. Si no disponemos de un aparato de TV en color, entonces sólo veremos diferentes tonos de gris. Sin embargo, los programas funcionarán perfectamente y, por tanto, podremos abordar su estudio sin problemas.

15.3.1 Características del color

El color es originado por la interacción de la luz con la retina del ojo. Por tanto, el color no es una propiedad de los objetos.

La sensación de color que experimenta un ser humano viene influida por un componente neurofisiológico (activación de ciertos mecanismos cerebrales) y un componente físico que depende de las características de la luz. Estas características son tres: el tono o longitud de onda, la saturación y la luminosidad o brillo.

Tono

Saturación

Brillo

El *tono* o *longitud* de onda se refiere a aquella característica que nos permite clasificar un color rojo, verde, azul, etc.

La *saturación* (llamada también pureza) nos describe el grado en que un color se separa de un gris neutro y se acerca a un color puro. Estas dos características constituyen conjuntamente la *cromaticidad* de la luz. La *luminosidad* o *brillo* se evalúa con referencia a una escala de grises que varían desde el blanco (máximo brillo) hasta el negro (mínimo brillo) pasando por toda la gama intermedia. Entonces, el brillo de una sensación de color se define (haciendo abstracción del tono y la saturación) como equivalente a la sensación producida por un elemento de la escala de grises.

Mezclas sustractivas

La luz que vemos normalmente, tanto la producida por medios artificiales de iluminación como la procedente del Sol, está formada por mezclas de colores. La mezcla puede ser de tipo aditivo o de tipo sustractivo. El hecho de que sean posibles mezclas sustractivas se explica por la existencia de colores complementarios. Este fenómeno es fácil de comprobar si miramos una luz roja a través de un vidrio o plástico azul. Puesto que el azul es completamentario del rojo, tendremos la sensación de que la luz roja está apagada o, al menos, con una intensidad muy reducida según sea la pureza de los colores con que efectuamos la prueba.

Mezclas aditivas

Mediante la mezcla aditiva o sustractiva de tres colores en proporciones diversas se pueden obtener todos los colores existentes en la realidad. Conviene resaltar que el ojo humano no percibe la mezcla como tal sino como un color homogéneo. En principio, se pueden elegir tres colores cualesquiera para formar la base de un sistema de colores. Este principio es el que se aplica en la televisión en color. Los puntos de imagen están formados por tres elementos luminiscentes cada uno de un color base. Sin embargo, hay un problema. Debido a la propia naturaleza del funcionamiento de una pantalla de TV, sólo es posible obtener mezclas aditivas. Por tanto, ya no es válida la afirmación de que con tres colores cualesquiera se puedan obtener todos los demás. Afortunadamente, los colores rojo, verde y azul permiten obtener una gama muy amplia de colores con sólo mezclas aditivas. Por esta razón, estos tres colores se suelen denominar *colores primarios* y son los utilizados en la televisión en color.

Colores primarios

Los ordenadores que disponen de gráficos en color hacen uso, lógicamente, de este mismo principio. En el caso de gráficos en blanco y negro se memorizaba el estado de activación de cada pixel, es decir, si se trataba de un punto luminoso (activado) o negro (desactivado). Si el gráfico es en color, será necesario almacenar una información triple para cada pixel, puesto que ya hemos dicho que están compuestos por tres puntos luminiscentes correspondientes a cada color base.

En general, los ordenadores suelen imponer algunas limitaciones en la utilización de colores. Veamos por qué. Según lo que hemos afirmado anteriormente, un color se formará por la mezcla en distintas proporciones de los colores básicos. Por tanto, será necesario guardar tres valores numéricos distintos para cada pixel. Estos valores contendrán las proporciones de cada color en la mezcla. Aun suponiendo que empleemos valores con pocas cifras, habrá que emplear al menos un byte u octeto para cada valor, o sea, tres bytes por pixel. Este gasto de memoria contrasta fuertemente con el empleado en los gráficos en B/N en los cuales sólo era necesario un bit para almacenar el estado de un pixel. En cambio, en los gráficos en color necesitaremos 24 bits (3 bytes) como mínimo para especificar úni-

camente el tono. A esto habría que añadir la información relativa a la saturación y al brillo. El gasto total de memoria es muy notable. Si suponemos una pantalla corriente de 100×200=20000 pixels, a 3 bytes por pixel resultan un total de 60 Kby de memoria empleada en almacenar la información de una pantalla.

Excepto para los ordenadores especializados en gráficos esta cantidad de memoria supone un costo excesivo para la mayoría de microordenadores, pues supondría tener que ampliar muchísimo su memoria central. Para solventar este problema se recurre a dos simplificaciones.

La primera de ellas se emplea incluso en pantallas gráficas de alta calidad y consiste en reservar un solo bit para almacenar la información referente a cada color base. Puesto que hay tres colores, se necesitarán 3 bits por pixel. Dado que un ordenador maneja siempre conjuntos de, al menos, 8 bits (1 byte), los 5 bits restantes se utilizan para almacenar información sobre el brillo y la saturación. Por consiguiente, el gasto de memoria por pixel queda reducido a 1 By. ¿Qué efecto tiene esta simplificación sobre la reproducción de los colores? Utilizando tres bits tendremos solamente 8 combinaciones distintas, por lo que podremos reproducir 8 colores incluyendo el blanco y el negro. Estos colores se especifican en la tabla de la figura 7.

Obsérvese que los colores base (rojo, azul y verde) tienen un solo bit activado. El blanco corresponde a todos los bits activados y el negro, como era de esperar, corresponde a todos los bits desactivados. Los demás colores se obtienen por mezcla. Por ejemplo, el amarillo se obtiene mezclando el rojo (bit 2) y el verde (bit 3). El color azul claro (mezcla de azul y verde) recibe el nombre técnico de cyan. La tabla de la figura 7 constituye la paleta de colores que disponemos para realizar los gráficos.

Aun habiendo reducido el almacenamiento a 1 By por pixel, se precisan unos 20 Kby para almacenar una pantalla en color completa. Para la mayoría de microordenadores esto resulta todavía excesivo y entonces se recurre a la segunda simplificación. En lugar de atribuir un color a cada pixel, se atribuye un color a un grupo de pixels. Este grupo de pixels suele ser el que forma el recuadro donde se inscribe un carácter (ver el tema dedicado a caracteres gráficos). De esta forma, si un pixel adquiere un color, todos los demás pixels pertenecientes al mismo recuadro adquirirán automáticamente el mismo color, puesto que la memorización es única para todos ellos. De esta forma se logra reducir sustancialmente el gasto de memoria.

Así como la primera simplificación tenía el efecto de reducir todos los colores de la naturaleza a únicamente 8 posibilidades, la segunda simplificación obliga a que la frontera entre zonas distintamente coloreadas coincida con la separación entre caracteres de la pantalla.

15.3.2 Instrucciones para el manejo del color

Como ya vimos en los gráficos en B/N, para trazar una línea en la pantalla hay que activar los pixels con el color inverso al fondo de la pantalla. El color del fondo se denomina también color del papel. El color de las líneas se denomina, a veces, color de la tinta. En los gráficos en B/N sólo teníamos dos posibilidades, normal o inverso. Por el contrario, en los gráficos en color existen más posibilidades.

Figura 16. Tabla de código de colores en binario y decimal.

NUM	BINARIO	COLOR
0	000	NEGRO
1	001	AZUL
2	010	ROJO
3	011	PURPURA o MAGENTA
4	100	VERDE
5	101	AZUL CLARO
6	110	AMARILLO
7	111	BLANCO

Desafortunadamente, al igual que para el resto de operaciones gráficas, no existen unas instrucciones normalizadas en BASIC para seleccionar el color de fondo y el de la tinta. Lo que sí se cumple para la gran mayoría de versiones del BASIC, es que los colores se codifican asignándoles a cada uno un número. Además, este número suele coincidir con el expuesto en la tabla de la figura 16. Para poder realizar programas utilizando los colores, será necesario que adoptemos una instrucción para especificarlos.

En muchos ordenadores esta instrucción es la siguiente:

```
COLOR n,m
```

Instrucción COLOR

en donde n es el color de la tinta y m es el color del fondo. La palabra COLOR es idéntica en español y en inglés. Otros ordenadores tienen instrucciones separadas para indicar el color del papel y de la tinta. Aunque insistimos que la instrucción COLOR no forma parte del BASIC estándar, la adoptaremos por ser la de uso más extendido. En el capítulo «Prácticas con el microordenador» hemos visto las instrucciones reales que operan en nuestro ordenador.

Siguiendo con nuestra explicación, la instrucción:

```
COLOR 0,7
```

significa que usamos líneas negras sobre fondo claro. Asimismo, la instrucción:

```
COLOR 7,0
```

significa que las líneas son blancas sobre fondo negro.

Lógicamente, si los dos números son iguales, no se podrán realizar dibujos, puesto que el color de la tinta coincidirá con el del papel. Como ya conocemos, esta propiedad se aprovecha precisamente para borrar líneas.

Las instrucciones para operar con el color no sólo afectan a las operaciones gráficas, sino que también tienen efecto sobre la impresión de los caracteres normales. Cuando se imprime un carácter, se entiende que el color del papel es el correspondiente al recuadro donde se inscribe el carácter. Lógicamente, el color de la tinta es el color con que se dibuja la letra o símbolo. Vemos, pues, que al imprimir un carácter se tienen en cuenta los dos colores a la vez. En cambio, al trazar una línea gráfica, sólo se tiene en cuenta el color de la tinta. Por tanto es posible imprimir caracteres contiguos (por ejemplo, formando una palabra) que no sólo tengan color distinto, sino que además tengan fondos también distintos.

Un hecho que conviene no olvidar es que cuando se cambia el color de la tinta esta instrucción no tiene efecto sobre las líneas ya trazadas. Su efecto se manifiesta únicamente para las líneas, puntos o caracteres que se traten con posterioridad. Lo mismo ocurre con el color de fondo. Para cambiar el color de toda la pantalla, por ejemplo, dejarla de color verde, utilizaremos las instrucciones:

```
COLOR 0,4 : CLS
```

Al efectuar un borrado de pantalla (instrucción CLS) la pantalla adquiere el color fijado en la instrucción COLOR. Lógicamente esta operación implica la pérdida de cualquier dibujo o texto existente en aquel momento. Esto nos hace ver que la selección del fondo de la pantalla será la primera operación que deberá realizar un programa gráfico, pues de lo contrario nos será imposible cambiar el color del papel sin perder los dibujos o textos escritos.

Imprimiendo un espacio en blanco obtenemos un recuadro que tiene únicamente el color del fondo. Imprimiendo varios de ellos podemos construir una zona de pantalla con un color de fondo distinto del resto. El siguiente programa aplica este hecho para obtener diversas zonas coloreadas en la pantalla. Además nos servirá para practicar con los códigos de los colores.

Programa de «Bandas coloreadas»

```
10 REM Bandas coloreadas
20 FOR I=1 TO 7
30   COLOR 0,I
40   FOR J=1 TO 96
50     PRINT " ";
60     NEXT J
70   NEXT I
```

El programa consta de un bucle principal que va desde la línea 20 hasta la 70. La variable I va desde 1 hasta 7 y corresponde a los códigos de color de cada zona (7 en total). En la línea 30 se selecciona el color de fondo. Puesto que pretendemos colorear zonas sin dibujar nada, no hace falta cambiar el color de la tinta. Por esta razón, la variable I sólo se emplea en la segunda parte de la instrucción COLOR. A continuación, se imprime un conjunto de espacios en blanco para colorear la zona. Para evitar que estos espacios en blanco se impriman en vertical y se produzca el fenómeno de «scroll» (desplazamiento hacia arriba del contenido de la pantalla) se coloca un punto y coma (;) al final de la instrucción PRINT de la línea 50. Al ejecutar el programa, aparecerán siete bandas cada una de ellas con un color de la tabla de la figura 7 (excluyendo el negro).

Por el momento podemos controlar una de las características del color, el tono o longitud de onda. El control de la saturación sólo es posible en pantallas gráficas especializadas, por lo que nosotros no lo tendremos en cuenta. En cambio, en la mayoría de ordenadores es posible controlar la tercera característica, el brillo o luminosidad. La instrucción empleada tiene la siguiente sintaxis general:

```
BRIGHT n
```

Instrucción BRIGHT

La palabra BRIGHT significa brillante en inglés. El número n es un indicador del grado de brillantez. En los ordenadores sencillos, sólo admite dos valores, cero y uno. En este caso el dibujo tendrá brillo con intensidad normal o doble intensidad. En otros ordenadores es posible obtener una graduación más precisa del brillo, por ejemplo de cero a nueve. Por razones de compatibilidad entre las distintas versiones de BASIC emplearemos siempre la forma simple de la instrucción BRIGHT.

Para ver el efecto del cambio de brillo sobre un color emplearemos el programa anterior añadiéndole la siguiente línea:

```
15 BRIGHT 0
```

Ejecutaremos el programa y observaremos con detenimiento la luminosidad del color. A continuación cambiaremos el 0 por el 1, es decir:

```
15 BRIGHT 1
```

y ejecutaremos de nuevo el programa. Comprobaremos de este modo la diferencia de brillo con el caso anterior.

La idea de dibujar bandas coloreadas a base de imprimir espacios en blanco tiene más aplicaciones. A continuación expondremos un programa

que traza una serie de bandas horizontales y verticales que se cruzan formando una especie de tejido. Para lograr este efecto utilizaremos la función AT que nos permite colocar el cursor en una posición determinada. El listado es el siguiente:

Programa de «Bandas cruzadas»

```
 10 REM Bandas cruzadas
 20 COLOR 0,1
 30 FOR I=0 TO 21 STEP 3
 40   PRINT AT(I,0);
 50   FOR J=1 TO 32: PRINT " ";: NEXT J
 60   NEXT I
 70 FOR I=2 TO 30 STEP 4
 80   COLOR 0,I/4
 90   FOR J=1 TO 21
100   IF INT(J/6)*6()J THEN PRINT AT(J,I);" ";
110   NEXT J
120   NEXT I
```

Escogemos el color 1 (el azul) para las bandas horizontales. El bucle que va desde la línea 30 hasta la 60 es el encargado de dibujarlas. La instrucción PRINT de la línea 40 utiliza la función AT para establecer el inicio de la banda. La banda completa se dibuja mediante la línea múltiple 50. Seguidamente se pasa a dibujar las líneas verticales. Cada una de ellas tendrá un color distinto, puesto que se emplea la variable I para seleccionar el color (línea 80). Se divide por 4 a fin de adaptar el margen de valores de I (de 2 a 32) al margen de códigos de color (de 0 a 7). No hay problema con los decimales, pues la instrucción COLOR toma siempre valores enteros. Para que el cruce entre las barras tenga forma de tejido es necesario que las bandas verticales crucen alternativamente por delante y por detrás de las bandas horizontales. De esto se encarga la instrucción de la línea 100. Cuando J es múltiplo de 3, entonces quiere decir que su posición coincide con una banda horizontal (obsérvese el valor de STEP de la línea 30). Si hay que pasar por encima de una banda y por debajo de la siguiente, entonces cuando J sea múltiplo de 6 no hay que dibujar a fin de preservar el color existente. La última barra vertical coincide con el color de la pantalla. Por eso, sólo puede verse cuando cruza una línea horizontal.

Al ejecutar el programa, la pantalla queda cubierta por un entramado de bandas coloreadas. Observamos que en los puntos donde hay superposición no se produce mezcla de colores, sino que simplemente el color dibujado en último lugar sustituye al antiguo. Esto es lógico puesto que ya señalamos que nuestra paleta de colores había quedado reducida a solamente 8 colores posibles.

Efectuamos ahora algunas pruebas con el color de tinta. El experimento más sencillo consiste en trazar una recta de un color determinado. Escribamos el siguiente programa:

```
10 COLOR 1,0
20 PUNTO(0,0)
30 LINEA(100,100)
```

Al escribir RUN, aparecerá en pantalla una línea de color azul. Un fenómeno que observaremos a continuación es que el color de la tinta ha quedado fijado a 1 y por tanto cualquier cosa que se imprima en pantalla se escribirá en ese color. Comprobémoslo efectuando un LIST. Esto nos enseña que al terminar el programa se debe restablecer el color original.

Pensemos que incluso puede darse el caso de que el color de la tinta coincida con el papel y entonces nos será imposible ver el listado de las instrucciones.

Importante: Cuando es un programa hacemos uso de instrucciones para cambiar el color de fondo o de tinta, al final del mismo debemos restablecer los colores iniciales.

De todas maneras, si nos olvidamos de hacerlo, emplearemos entonces la instrucción COLOR (o su equivalente) en modo inmediato.

El siguiente experimento que realizaremos será la comprobación de si nuestro ordenador memoriza el color para cada pixel por separado o bien memoriza conjuntos de pixels. El programa anterior lo modificaremos para que trace dos rectas de distinto color que se cruzan. El listado es:

```
10 COLOR 1,0
20 PUNTO(0,0)
30 LINEA(100,100)
40 COLOR 2,0
50 PUNTO(0,100)
60 PUNTO(100,0)
70 COLOR 0,7
```

El programa dibuja dos rectas con los colores 1 (azul) y 2 (rojo) respectivamente. Obsérvese que la última instrucción (línea 70) restaura los colores originales. Aquí se ha supuesto que usamos normalmente caracteres escritos en negro sobre fondo blanco. Si no es así, invertimos los valores de la instrucción COLOR.

Un detalle sobre el que nos fijaremos atentamente es el punto de cruce de las dos rectas. Si en la primera recta no se ve afectado el color de ninguno de sus puntos (excepto la intersección, claro está) entonces significa que el ordenador memoriza por separado el color de cada pixel. Si, por el contrario, varios pixels de la primera recta cercanos a la intersección adquieren el color de la segunda, entonces quiere decir que nuestro ordenador memoriza el color por grupos de pixels.



15.3.3 Aplicaciones

Barras en color

El primer caso práctico será la construcción de un diagrama de barras, en donde utilizaremos el color para destacar una barra respecto a las demás. En la primera lección de gráficos ya vimos un programa de este tipo, cuya estructura básica aprovecharemos ahora. El listado del programa es el siguiente:

Programa de «Diagrama de barras en color»

```
 10 REM Diagrama barras en color
 20 LET N=5: DIM A(N),C(N)
 30 FOR I=1 TO N: READ A(I),C(I): NEXT I
 40 LET M=A(1)
 50 FOR I=2 TO N
 60   IF A(I)>M THEN LET M=A(I)
 70   NEXT I
 80 LET E=80/M: LET D=160/(N+1)
 90 GOSUB 300
100 FOR I=1 TO N
110   LET P=I*D: LET L=A(I)*E
120   GOSUB 400
130   NEXT I
140 COLOR 0,7
150 STOP
160 DATA 10,2, 5,2, 4,4, 20,2, 16,2
300 REM Ejes
310 PUNTO(24,80)
320 LINEA(24,24)
330 LINEA(160,24)
340 RETURN
400 REM Barra
410 COLOR C(I),0
420 FOR J=24 TO L+24
430   PUNTO(P+24,J)
440   LINEA(P+48,J)
450   NEXT J
460 RETURN
```

Como sabemos, la estructura básica es la siguiente. En primer luga se leen los datos de las alturas de las barras (en el vector A) y de sus colores (en el vector C). En la instrucción DATA de la línea 160 se disponen estos datos de forma alternada, es decir que 10,2 significa altura 10 y colo 2. A continuación se determina el valor máximo de la lista (líneas 40 a 70).

Seguidamente se dibujan los ejes (subrutina 300) y empieza el bucle de dibujado de las barras.

Instrucción PAINT

La diferencia de este diagrama con el de la lección anterior es que las barras deben ser coloreadas. Algunos ordenadores disponen de la instruc

ción PAINT (paint significa pintar en inglés) que automáticamente rellena una figura cerrada con un color determinado. Si no disponemos de esta instrucción, se puede construir fácilmente un segmento de programa que efectúe este cometido. En la subrutina 400 se muestra un ejemplo. El método se basa en trazar segmentos rectilíneos del color escogido, uno encima de otro de modo que se construya el rectángulo. Puesto que no existe separación entre pixels, no se apreciará tampoco separación entre los diversos segmentos. En consecuencia, obtendremos una superficie coloreada continua.

Todas las barras son del mismo color (rojo), excepto la tercera, que tiene color verde, destacando así sobre las demás. La última instrucción antes del STOP sirve para restaurar los colores iniciales.

El segundo ejemplo es un programa para dibujar una bandera. En este caso se ha escogido la bandera de la Cruz Roja por la simplicidad de sus colores. El listado es el siguiente:

Programa de «Dibujo de bandera»

```
 10 REM Bandera
 20 COLOR 0,7: CLS
 30 COLOR 0,2
 40 FOR I=4 TO 8
 50   PRINT AT(14,I);"     "
 60   NEXT I
 70 FOR I=9 TO 13
 80   PRINT AT(9,I);
 90   FOR J=1 TO 15
100     PRINT " ";
110     NEXT J
120   NEXT I
130 FOR I=14 TO 18
140   PRINT AT(14,I);"     "
150   NEXT I
160 COLOR 0,7
```

La primera operación, realizada por las instruccioes 20 y 30, selecciona los colores. En primer lugar coloca la pantalla con fondo blanco y a continuación selecciona el color rojo (número 2). Esta instrucción no tendrá efecto hasta que no escribamos algo en la pantalla. El resto del programa está formado por tres bucles. El primero va desde la línea 40 a la 60, el segundo de la 70 a la 120 y el tercero de 130 a la 150. Cada uno de estos bucles construye un trozo de la cruz. En las líneas 50 y 140 las comillas engloban 5 espacios en blanco. Por el contrario, en la línea 100 sólo hay un espacio. Se utiliza la función AT para colocar el cursor en la posición adecuada.

Un punto importante es la utilización de colores para representar superficies vistas en perspectiva. El procedimiento consiste en rellenar (o pintar) la superficie de un determinado color, en lugar de utilizar el procedimiento de la retícula. Este método tiene la ventaja de que nos permite dar

la sensación de que hay caras que están iluminadas y otras que están en la sombra. Esto se realiza empleando colores claros y oscuros. De esta forma se consigue dar una mayor sensación de profundidad.

Representación de objetos sólidos

Como ilustración de este método representaremos un objeto sólido. Se ha escogido un prisma de sección cuadrada porque es bastante sencillo de construir. Se ha dibujado en perspectiva y se ha supuesto que la iluminación es frontal.

Programa de «Objetos sólidos»

```
 10 REM Objetos solidos
 20 FOR I=16 TO 80
 30   COLOR 5,0
 40   PUNTO(80,I)
 50   LINEA(104,I)
 60   COLOR 1,0
 70   LINEA(112,I+8)
 80   NEXT I
 90 FOR I=80 TO 104
100   PUNTO(I,80)
110   LINEA(I+8,88)
120   NEXT I
130 COLOR 0,7
```

El coloreado de las superficies se efectúa a base de superponer segmentos. Se ha escogido el color azul claro (también llamado cyan) para las superficies encaradas a la fuente de iluminación y el color azul oscuro (número 1) para las superficies no iluminadas directamente.

La primera parte del programa, desde la línea 20 hasta la 80, se dibujan la cara frontal (color claro) y la lateral derecha (color oscuro). La segunda parte, que comprende desde las líneas 90 hasta la 120, dibuja la superficie superior del prisma.

Al ejecutar el programa, aparece en pantalla un prisma que da mayor sensación de solidez que si se hubiera empleado el método de la retícula.



15.4 MEMORIZACION PANTALLA

A menudo, durante la realización de gráficos surgen dos necesidades. La primera de ellas es averiguar si una posición determinada de la pantalla (un pixel) está activado o desactivado. La segunda es la memorización de un gráfico, a fin de poder representarlo posteriormente sin necesidad de efectuar todas las operaciones necesarias para construirlo. Ya conocemos por experiencia que ciertos gráficos implican un tiempo de cálculo muy elevado. Ciertamente, toda la información gráfica de la pantalla se encuentra almacenada en una zona de la memoria central del ordenador. Sin em-

bargo, esta memoria es única y, por tanto, sucesivos dibujos destruyen los anteriores. De lo que se trata es de guardar el contenido de esta zona de memoria en otra zona de la memoria central, como por ejemplo un conjunto dimensionado, o bien trasladarla a una unidad de almacenamiento auxiliar como una cinta magnética.

Los diversos modelos de ordenador se diferencian muchísimo entre sí en lo que se refiere a las posibilidades gráficas apuntadas aquí. Por esta razón, sólo expondremos un programa de este tipo. En el capítulo de «Prácticas con el microordenador» correspondiente a este tema ha visto ya una explicación más detallada sobre estos puntos.

El siguiente programa es un ejemplo de la consulta del estado de un pixel. Se trata de dibujar un punto móvil en la pantalla que cuando encuentre un obstáculo rebota en sentido contrario. Antes de empezar, definiremos la función:

ESTADO(X,Y)

que nos dará el estado de activación de pixel situado en las coordenadas (X,Y). Si el resultado es 1, el pixel está activado. Si el resultado es cero, el pixel está desactivado. Lógicamente, esta instrucción es convencional, al igual que PUNTO y LINEA. En el capítulo de «Prácticas con el microordenador» ha visto las instrucciones que funcionan en nuestro ordenador. El listado del programa es el siguiente:

Programa de «Punto móvil»

```
 10 REM Punto movil
 20 INPUT "POSICION:",P
 30 PUNTO(P,1)
 40 LINEA(1,100)
 50 PUNTO(1,50)
 60 FOR X=2 TO 200
 70   IF ESTADO(X,50)=1 THEN GOTO 110
 80   PUNTO(X-1,50),I
 90   PUNTO(X,50)
100   NEXT X
110 M=X-1
120 FOR X=M TO 1 STEP -1
130   PUNTO(X+1,50),I
140   PUNTO(X,50)
150   NEXT X
```

Las primeras instrucciones (de la línea 20 a la 40) sirven para colocar una recta vertical en una posición elegida por el usuario. Esta recta será el obstáculo donde rebotará el punto móvil. A continuación se dibuja el punto inicial y empieza el proceso de movimiento. El movimiento se consigue, como siempre, borrando la posición actual y dibujándolo en la posición adyacente. Sin embargo, en este caso hay una diferencia. Antes de efectuar

el movimiento, se consulta el estado del pixel donde nos dirigimos (línea 70). Si el pixel está activado significa que aquella posición está ocupada y por tanto se debe producir el rebote. Por consiguiente, se pasa control a la línea 110, en donde empieza el bucle de movimiento del punto en sentido contrario, es decir, hacia la izquierda de la pantalla.



15.5 PRODUCCION DE SONIDO

La parte que sigue vamos a dedicarla a una nueva posibilidad de nuestro ordenador: producir sonido.

Para ello es necesario que el ordenador posea el dispositivo adecuado, que incluye un altavoz. Si su ordenador tiene esta posibilidad ya lo sabe por el capítulo de «Prácticas con el microordenador». Si no posee el dispositivo de producir sonido, no será necesario que prosiga la lectura de estos apartados. Por el contrario, si su ordenador tiene esta posibilidad, encontrará a continuación la descripción general del procedimiento a seguir. En este caso es difícil dar una descripción de la instrucción en concreto, dado que no existen instrucciones estándar para estos procesos. En cada caso deberá consultar en el capítulo de «Prácticas con el microordenador» la sintaxis correcta, que es lo que acaba de estudiar.

La orden fundamental para producir sonido está compuesta de tres elementos. El primer elemento en este caso es una palabra clave, que corresponde a la instrucción de producir sonido. Junto a esta palabra se escriben dos cualificadores, que constituyen el segundo y tercer elemento de la instrucción, e indican respectivamente la duración del sonido y el tono del mismo.

Duración y tono del sonido

Instrucción BEEP

Estos dos elementos se pueden indicar directamente mediante números, o bien indicarlos utilizando variables, e incluso expresiones complejas, siempre que sigan las normas de escritura de las expresiones en BASIC. En este caso, el resultado de la evaluación de las mismas indicará el valor a utilizar. Para ver un ejemplo de cómo se escribiría este comando, supondremos que la orden de producir sonido se da mediante la palabra BEEP, que es efectivamente utilizada por varios ordenadores. Así, podríamos escribir por ejemplo:

```
BEEP 1,3
```

La forma general de la instrucción para una máquina cualquiera sería:

Orden de sonido Duración, Tono

Los valores concretos a utilizar para duración y tono dependen también de la máquina; para conocer el significado completo de cada valor en su

ordenador deberá consultar el correspondiente capítulo de «Prácticas con el microordenador».

Para poder explicar de una forma concreta el funcionamiento de la instrucción, vamos a ver ejemplos que, aunque no son válidos para todos los ordenadores —hemos dicho que esta instrucción no es estándar—, nos resultarán útiles para comprender mejor la utilización de la misma, así como de los parámetros que la acompañan.

Hemos escrito pues un ejemplo de esta forma:

```
BEEP 1,3
```

Veamos qué significa. El primer número generalmente corresponde al tiempo —en segundo— que se mantiene el sonido. Así, para el ejemplo que hemos escrito, la duración del sonido sería de un segundo.

Por otra parte, el segundo valor corresponde al tono del sonido a reproducir. En principio, y de una forma simplificada, podemos decir que el tono corresponde a la nota que se desea reproducir.

Para componer e interpretar música con su ordenador, no es necesario que usted tenga conocimientos musicales de ningún tipo.

Manejando la instrucción de forma adecuada y a su gusto, podrá producir la música que usted desee.

Así, para producir los distintos sonidos, se escribirán distintos valores en la posición del tono. Estos valores siguen un orden concreto. Generalmente, se indican escribiendo valores positivos o negativos, entre unos márgenes determinados. Los valores más pequeños corresponden a tonos más bajos, producirán sonidos más graves, mientras que al ir aumentando el valor del tono iremos produciendo sonidos más agudos, correspondientes a tonos más altos.

Generalmente el valor cero corresponde a un tono intermedio, que se toma como base, los sucesivos valores positivos darán tonos sucesivamente más altos, mientras que los valores inferiores al cero (evidentemente valores negativos) darán tonos sucesivamente más graves.

Para comprender mejor este mecanismo, imagínese el teclado de un piano (Fig. 17). En él se sitúa el valor cero en una posición dada, supongamos por ejemplo, que esté en la nota DO del centro del teclado, tal como indica la figura. Para conseguir la nota siguiente (correspondiente al sonido que se obtendría pulsando la siguiente tecla blanca), escribiríamos:

```
BEEP 1,2
```

Para reproducir la siguiente, deberemos escribir

```
BEEP 1,4
```

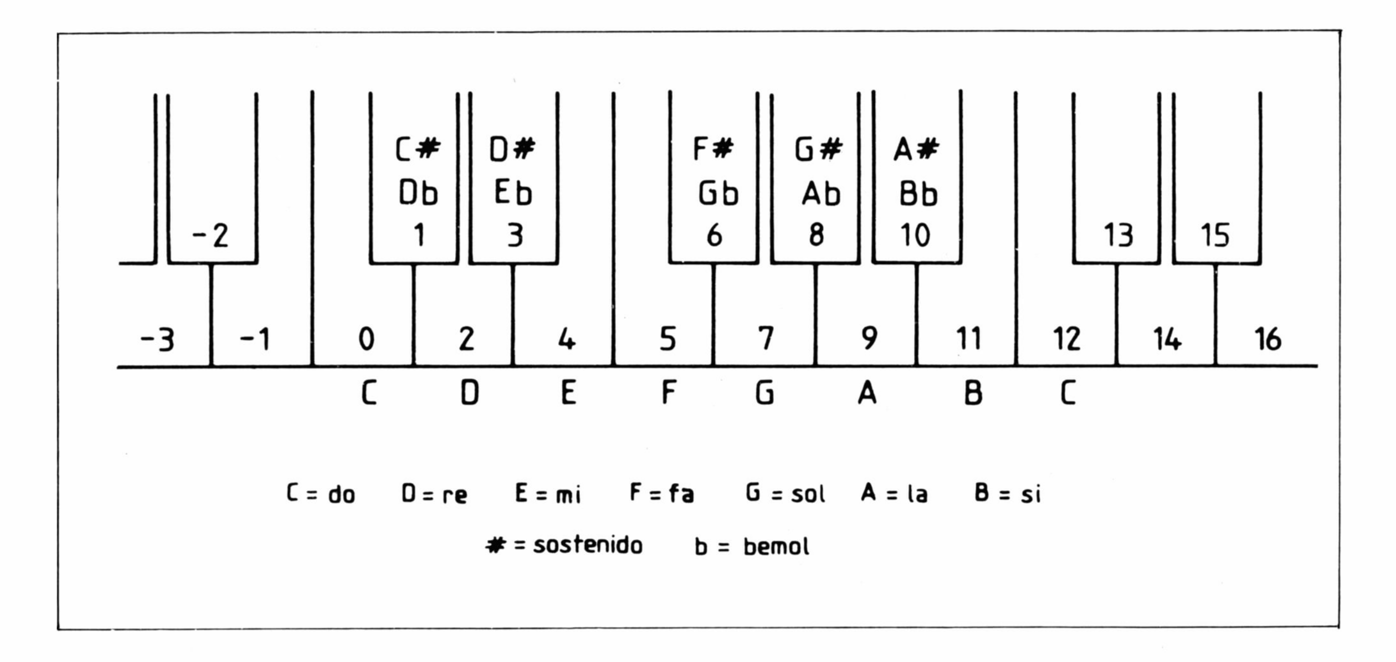


Figura 17. Situación de las notas musicales respecto al teclado.

Los valores impares corresponderían entonces al sonido intermedio, el de la tecla negra situada entre las dos blancas. La diferencia entre dos teclas blancas es de un tono, y entre una blanca y la negra consecutiva, de un semitono. Así, los sucesivos números crecientes darán sonidos que se diferenciarán sucesivamente en un semitono, de la misma forma que si vamos pulsando las teclas del piano, incluidas las negras, desde el centro hacia la derecha. Por tanto, para reproducir el sonido del piano, escribiríamos:

```
BEEP 1,0
BEEP 1,1
BEEP 1,2
BEEP 1,3
...
```

Para reproducir las notas inferiores al valor central deberíamos escribir:

```
BEEP 1,0
BEEP 1,-1
BEEP 1,-2
BEEP 1,-3
...
```

Aunque en su ordenador no sean exactamente éstos los valores que se utilicen, el criterio seguido para producir los sonidos suele ser muy semejante; consultando el capítulo de «Prácticas con el microordenador», correspondiente a este tema, encontrará el funcionamiento exacto de la instrucción.

Por otra parte, y en general, los valores de tiempo y tono no tienen por qué ser enteros; podemos pues utilizar valores decimales para ambos, lo cual nos permite ajustar al máximo —si tenemos el oído suficientemente educado como para hacerlo— una melodía cualquiera al sonido real de la misma. La única limitación de nuestra máquina es que sólo puede producir un sonido cada vez.

Por otra parte, en cuanto al intervalo de sonido que podemos producir, generalmente se puede abarcar todo el espectro de sonidos perceptibles por el oído humano, variando los valores del tono. En todo caso deberá comprobarlo en su máquina, pues los valores de tiempo y tono a utilizar se deben encontrar entre un máximo y un mínimo, y si se utiliza algún valor fuera de este margen, la máquina nos dará un error.

RESUMEN

El color no es una propiedad de los objetos sino que depende del observador y de las características de la luz. Estas características son tres, el tono o longitud de onda, la saturación y el brillo. El tono establece el tipo de color. La saturación equivale a la pureza del color, es decir, al grado de separación del gris neutro. El brillo es la luminosidad de un color.

En general, la luz está formada por una mezcla de colores. Las mezclas pueden ser aditivas o sustractivas. Mediante este tipo de mezcla y utilizando tres colores se pueden reproducir todos los colores existentes en la realidad. En un aparato de televisión, las mezclas son siempre aditivas; por tanto, los tres colores primarios deben ser el rojo, verde y azul, puesto que son los que dan una gama de colores más amplia.

En los gráficos en color, la información que debe almacenarse para pixel es muy grande. Por tanto, se recurre a simplificaciones a fin de ahorrar memoria. Esto trae como consecuencia una reducción en el número de colores utilizables (denominado paleta de colores) y en los grados de saturación y brillo representados.

En la pantalla existen dos colores distintos. El color de fondo o del papel y el color de las líneas o de la tinta. Cada uno de ellos puede tener cualquier valor de los admitidos e incluso pueden coincidir. Cuando un programa modifique uno de estos colores, al final debe restaurar los colores iniciales, pues, de lo contrario, la pantalla quedará con el color empleado por última vez.

Cuando se representan objetos sólidos en color, se suele suponer la existencia de una fuente de iluminación. Entonces las caras iluminadas se trazan con colores más claros o con brillo más intenso.

El estado de una pantalla se puede memorizar parcial o globalmente. La memorización global nos permite reproducir una pantalla entera sin tener que recalcularla. Por otra parte, los ordenadores suelen incorporar una función para consultar el estado de un pixel.

Para producir sonido, la orden fundamental suele ser la instrucción BEEP seguida de dos números que indican el tono del sonido y su duración.

EJERCICIOS DE AUTOCOMPROBACION

Completar las frases siguientes:

26. El tono o longitud de onda y la .... constituyen la cromaticidad de la luz.

27. El color de fondo se denomina también color ....

28. La sensación de color depende del observador y de las características de la ....

29. La selección del color de fondo de la pantalla será la .... operación que realizará un programa.

30. Para construir una zona coloreada en la pantalla imprimiremos varios ....

31. Mediante la utilización de colores claros y oscuros se simula la existencia de una ....

32. La realización de un nuevo dibujo .... cualquier dibujo anterior.

33. Un pixel está .... si tiene el color de la tinta.

34. Si la memorización de los colores se efectúa por grupos, la separación entre dos zonas de distinto color deberá coincidir con la separación entre..................................

35. Para reproducir sonido con un ordenador necesitamos controlar al menos el .... y la duración.

Encierre en un círculo la respuesta correcta: La (S) si es verdadera y la (N) si es falsa.

36. Utilizando solamente mezclas aditivas, se pueden elegir tres colores cualesquiera para formar una base y reproducir todos los demás .... S N

37. Los colores de fondo y de la tinta no pueden coincidir ...... S N

38. Para rellenar una figura con un determinado color, trazaremos una serie de segmentos adyacentes de un lado a otro de la figura .... S N

39. La memorización de un gráfico nos permite ahorrar instrucciones y tiempo de cálculo en un programa que utilice dicho gráfico .... S N

40. Los tonos más altos producirán sonidos más graves .... S N

SOLUCIONES DE LOS EJERCICIOS DE AUTOCOMPROBACION

Capítulo 13

1. estructura.
2. bloque.
3. programa.
4. instrucción.
5. subrutina.
6. refinamiento.
7. función.
8. secuencial.
9. un bloque.
10. múltiple.
11. d.
12. b.
13. a.
14. a.
15. c.
16. b.
17. a.
18. a.
19. c.
20. c.
21. repetitiva, iterativa.
22. sabe.
23. iterativa.
24. repetitiva.
25. dos.
26. subbloque B.
27. MIENTRAS.
28. HASTA.
29. ordinogramas.
30. rectángulos.
31. d.
32. c.
33. b.
34. c.
35. c.
36. a.
37. c.
38. b.
39. a.
40. a.

Capítulo 14

1. 127. (Los demás son caracteres estándar.)
2. CHR$.
3. Pixel.
4. Resolución.
5. Bit. (Cada pixel se almacena en un bit de la memoria central.)
6. Abscisas.
7. Dos.
8. Inferior izquierdo.
9. Unidades.
10. Primitivas.
11. Punto actual.
12. Pixels.
13. Pendiente.
14. Mayor. (Mayor resolución indica pixels más pequeños y por tanto mejor calidad.)
15. Primitivas.
16. Radianes.
17. –1.
18. Crece.
19. El propio ángulo.
20. Inversa.
21. a). (Todos los caracteres ocupan un recuadro por definición.)
22. c).
23. d). (Cada modelo de ordenador realiza su propia subdivisión.)
24. c).
25. c).
26. b).
27. a). (El pixel es la unidad básica en una pantalla gráfica.)
28. c).
29. d). (La pendiente de una recta no puede ser errónea.)
30. a).
31. Circunferencia o círculo.

32. Semirradios.
33. Creciente.
34. Escala.
35. Angulo.
36. COS.
37. X–Y. (En el eje de abscisas se colocarán los días y en el eje de ordenadas la cotización.)
38. Inverso.
39. Desactivación.
40. Borrar.
41. b). (De esta forma calculamos fácilmente los vértices.)
42. d).
43. d). (Al cambiar el centro, el sector quedará desplazado respecto a los demás.)
44. b). (El modo superposición se invierte el estado inicial.)
45. a).
46. S.
47. S. (No es posible dibujar un sector con ángulo negativo.)
48. N. (Se repreentaría en un diagrama X–Y.)
49. N. (Depende de la precisión con que se trace la circunferencia.)
50. S. (Uno para cada eje.)

Capítulo 15

1. Vértices.
2. Escalado.
3. Izquierda
4. Reducir. (Para reducir, el factor de escala será menor que la unidad.)
5. Deformada. (Se pierde la proporción original entre anchura y altura.)
6. Inferior.
7. Orientación. (La posición debe coincidir con la original. Solamente se produce un giro.)
8. El mismo sentido. (Aunque los ángulos se miden en sentido contrario, la operación de rotación funciona en el otro sentido.)
9. Superficies o caras.
10. Transparencia.
11. Perspectiva. (Esta operación es obligada para representar un objeto tridimensional sobre un plano, la pantalla.)
12. Superficie.
13. Mayor. (Ocurre lo mismo que con la resolución.)
14. Puntos. (De esta forma sabemos si un punto queda detrás o delante de otro.)
15. Mayor. (Al aumentar la dimensión aumenta también el número de cálculos.)
16. a). (El primer punto se traza de forma distinta al resto.)
17. b).
18. c). (La instrucción CLS borra toda la pantalla y no sólo a la figura.)
19. d). (El traslado a (0,0) es necesario para que la figura no cambie de posición.)
20. c). (El eje de giro, o centro de la figura, debe pasar por el origen de coordenadas.)
21. b). (Al cambiar la inclinación de las líneas se obtiene la sensación de un cambio de plano o cara.)
22. a). (En un objeto opaco, las caras posteriores no son visibles.)
23. a).
24. a). (Es más fácil de programar que el sombreado.)
25. d). (Ver si un punto está oculto es más fácil que hacer lo mismo para una recta.)
26. Saturación.
27. Del papel.
28. Luz.
29. Primera. (Esto es debido a que necesitamos borrar la pantalla.)
30. Espacios en blanco.
31. Fuente de iluminación.
32. Destruye o borra.
33. Activado.
34. Caracteres.
35. Tono.
36. N. (Para conseguirlo se necesitan mezclas aditivas y sustractivas.)
37. N. (Precisamente la coincidencia se utiliza para borrar.)
38. S.
39. S.
40. N. (Producirán sonidos más agudos.)

Parte II

PRACTICAS CON EL ORDENADOR

Capítulo 13

ESQUEMA DE CONTENIDO

13.1 OBSERVACIONES GENERALES

Le vamos a recordar algunas cuestiones que ya se han tratado y que son particularidades del ZX-Spectrum.

En primer lugar en varios programas de la lección del BASIC estándar se utiliza la instrucción END que no tiene el XS-Spectrum. En cualquier caso puede ser sustituida por un GO TO 9999. Ningún programa utiliza esta línea y sirve, por lo tanto, como final.

Por otra parte, en el apartado 13.3.2 se utiliza la función INKEY$. En el caso del ZX-Spectrum es mejor sustituir la línea 110 de este programa por

```
110 PAUSE 0 : A$=INKEY$
```

de esta manera se evitan repeticiones de las teclas.

En el apartado 13.4 hay dos versiones de los programas que buscan las iniciales del mes, también hay una en el apartado 13.4.2. En el programa se utiliza una tabla de caracteres para almacenar las iniciales de los meses. En el ZX-Spectrum es obligatorio definir el número de caracteres que tiene cada uno de los elementos de la tabla. Es necesario, por lo tanto, modificar la instrucción 20 que debe quedar como

```
20 DIM M$(12,3)
```

que define una tabla de 12 elementos y 3 caracteres cada uno.

En el programa de contar palabras del apartado 13.4.1, en sus dos versiones, se utiliza la función MID$ para extraer caracteres. Esta función no existe en el ZX-Spectrum, en su lugar hay que utilizar el operador de fragmentación. La línea a sustituir es la 2500 que está escrita como

```
2500 LET C$ = MID$(A$,I,1)
```

y debe quedar como

```
2500 LET C$ = A$(I TO I)
```

Finalmente las instrucciones que se mencionan en el apartado 13.3.3

ON....GO TO

ON....GO SUB

no se pueden utilizar en el ZX-Spectrum.

De todas maneras es muy conveniente que lea el texto y lo entienda, pues prácticamente todos los demás dialectos del BASIC la tienen y aunque no le sirva para el ZX-Spectrum se puede encontrar con la necesidad de adaptarla desde otro programa.

La técnica de adaptación es la descrita en el apartado 13.3.2, utilizando la cadena de IF para ir eliminando opciones.

13.1

13.2 PRACTICA 1. UNA CALCULADORA

La práctica que ahora va a comenzar es un poco larga. Por tanto realícela cuando tenga tiempo disponible, pues es posible que a la hora de introducir los programas cometa errores. Si esto ocurre, repáselos con tranquilidad y no tardará en descubrir los fallos. Por otra parte, le recomendamos mucha atención, pues es fácil el equivocarse. Le recomendamos que tenga a punto la grabadora, pues si tiene que cortar el estudio en un momento dado es mejor no perder la parte del programa que tenga ya tecleada.

La realización de una calculadora no parece un problema interesante ya que el BASIC dispone de la manera de hacer cálculos sin necesidad de recurrir a un programa especial.

El interés que presenta esta práctica es doble:

a) Por una parte, se aprende el funcionamiento de mecanismos que el propio BASIC utiliza para su funcionamiento interno.

b) Por la otra, se consigue una calculadora más ágil que el propio BASIC si se han de hacer muchos cálculos sencillos que no merece la pena dedicar un programa para realizarlos específicamente.

13.2.1 Definición del problema

Se trata de construir una calculadora que realice las cuatro operaciones aritméticas: sumar, restar, multiplicar y dividir, con la característica de memorizar resultados intermedios.

Además de estas cuatro operaciones ha de poseer una para finalizar el programa.

El aspecto más interesante a comentar de esta definición es la memorización de los resultados intermedios.

Normalmente en una calculadora si queremos hacer una multiplicación seguimos la secuencia siguiente de teclas:

1. Se teclea un número.
2. Se pulsa el signo de la multiplicación.
3. Se pulsa el otro número.
4. Se pulsa la tecla igual.

En el tipo de calculadora que desarrollaremos la secuencia es algo distinta

1. Se teclea el número.
2. Se pulsa la memorización.
3. Se pulsa otro número.
4. Se pulsa la tecla de la multiplicación.

En el ejemplo de multiplicar 3 por 4, las dos columnas describen esta secuencia de operaciones en cada una de las máquinas.

1. 3	3
2. *	<– –
3. 4	4
4. =	*

Como puede apreciar, la diferencia no es muy grande, pero es significativa. El orden de los números es el mismo, pero en la calculadora que desarrollamos se utiliza primero un símbolo desconocido en la mayoría de calculadoras. Para acabar la operación tecleando el signo de la operación aritmética.

¿Por qué se elige este segundo tipo de calculadora? La razón es que se pueden memorizar fácilmente resultados intermedios. Recuerde que en una expresión en BASIC los resultados intermedios son los que se deben calcular antes, porque están entre paréntesis.

Para mostrar de una manera más clara las diferencias entre una y otra calculadora realicemos el ejemplo siguiente:

$$3 * (4 + 2)$$

en el que intervienen paréntesis.

En las calculadoras normales debemos alterar el orden de evaluación para conseguir eficiencia, es decir la secuencia de teclas que utilizamos es

1. 4	4
2. +	4
3. 2	2
4. =	6
5. *	6
6. 3	3
7. =	18

(en la segunda columna es el resultado que aparece en la línea de pantalla de la calculadora).

El hecho importante aquí es que para realizar la operación tenemos que alterar el orden de las operaciones tal como aparecen en la expresión; es decir, hay que calcular primero 4+2 para luego poder multiplicar por 3.

Si se nos descubre poco a poco la expresión de izquierda a derecha no podemos saber cómo cacularla hasta saber toda la expresión.

Esto no ocurre en el tipo de calculadora que haremos. El símbolo raro (<– –) que hemos utilizado sirve para indicar que se debe memorizar el número.

El proceso de memorización se realiza sobre una hilera de memorias de tal manera que se va ocupando sucesivamente la primera, la segunda, la tercera y así sucesivamente. Cuando se memoriza un número nunca se escribe sobre una memoria ocupada, se escribe en la siguiente libre.

Cuando realizamos operaciones es cuando se disminuye la memoria.

Veamos el mecanismo detallado que sigue el tipo de máquina que deseamos.

Los pasos son los siguientes:

1. Se nos descubre el 3. Se teclea y se memoriza en la primera posición, pues es la primera memoria no ocupada.

2. Se nos descubre el signo por. De momento no hacemos nada.

3. Se nos descubre el paréntesis. El próximo número hay que memorizarlo.

4. Se nos descubre el 4. Se teclea y se memoriza en la segunda memoria ya que es la primera no ocupada. Ahora en la memoria primera tenemos un 3 y en la segunda memoria tenemos un 4.

5. Se nos descubre el signo más. De momento no hacemos nada.

6. Se nos descubre el 2. Se teclea y se memoriza en la memoria tercera. Tenemos ahora tres números en las tres primeras memorias, el 3, el 4 y el 2.

7. Se nos descubre el paréntesis de cerrar. Se debe encontrar el resultado de la operación intermedia. ¿Qué signo aritmético está pendiente?; el signo más, que es el último que se ha encontrado.

8. Se aplica el signo más sumando las dos últimas memorias (la dos y la tres) y el resultado se deja en la memoria de más bajo orden, es decir, sobre la memoria dos. De tal manera que después de realizar esta operación sólo tenemos dos memorias ocupadas, la uno y la dos, con los números 3 y 6 respectivamente.

9. Se descubre a continuación que se ha terminado la expresión y queda pendiente el signo aritmético por.

10. Se aplica la multiplicación entre las dos últimas memorias ocupadas, la uno y la dos, y el resultado se deja en la más baja, en este caso la uno. Que según el ejemplo queda con un 18.

El resultado queda en la memoria uno.

Una manera abreviada de expresar este mecanismo es mediante la secuencia

3 <– – 4 <– – 2 <– – + *

en donde el símbolo, <– –, significa la introducción del número en la memoria encima de la última ocupada, o lo que es lo mismo, en la primera libre.

Se demuestra que en una máquina de este tipo no es necesario tener en cuenta los paréntesis. La memorización de un resultado sustituye a los paréntesis.

Por otra parte, los operadores, cuando se aplican, se hacen sobre las dos últimas memorias ocupadas, el resultado se deja en la memoria más baja.

A la notación de las expresiones mediante esta técnica se la denomina *notación polaca reversa*, porque el orden de los operadores aparece en un orden «invertido» respecto a la escritura normal de expresiones. Se ha situado entre comillas el orden invertido porque sólo se invierte cuando en las otras expresiones hay paréntesis.

La principal herramienta de esta máquina de calcular es la memoria que dispone que se ocupa a medida que memorizamos números y se desocupa a medida que realizamos operaciones.

A este tipo de memoria se la denomina pila, a semejanza de una pila de platos o de papeles.

Cuando nos llega un número lo apuntamos en un papel y lo depositamos encima de la pila de papeles que ya tenemos.

Cuando hay que realizar una operación tomamos los dos papeles de arriba, realizamos la operación, la apuntamos sobre un papel (se aprovecha uno de los antiguos, borrando y anotando el valor del resultado) que se deposita encima de la pila otra vez. Se desechan los papeles utilizados para la operación.

Para comprender bien el mecanismo vamos a hacer un pequeño programa prototipo para ver el funcionamiento.

13.2.2 La calculadora prototipo

Las ideas claves de este programa son:

a) Una tabla de número se utiliza como memoria de pila.

b) Para saber cuál es el último elemento ocupado se tiene un número que inicialmente vale cero, que crece en una unidad cuando se memorice un número y decrece en una unidad cuando se realiza una operación.

c) Hay un bucle de órdenes que nos indica cuándo hay que memorizar u operar. Si la entrada es de memorización se pide a continuación el número.

d) Cuando la entrada sea alguno de los símbolos de los signos aritméticos se toma como indicación de operación. En cualquier otro caso se considera que se trata de un símbolo de memorización.

Para realizar este proceso se dispone del bucle que lee órdenes y las interpreta según el símbolo que se lea. Para tener un control estricto del contenido de la memoria de la pila se visualiza cada vez entre orden y orden.

Es necesario disponer de dos subrutinas, una que memorice y otra que opere, que son las dos acciones básicas que hemos mencionado.

El programa se inicia con la definición de la memoria de la pila y la inicialización del número que nos indica la última memoria ocupada. Cuando no hay ninguna se coloca a cero.

```
10 REM -- Inicializaciones
20 DIM p(10)
30 LET pp = 0
```

La variable *p* es la tabla que contiene la memoria de pila y *pp* la variable que indica qué elemento es el último de la memoria.

Se dimensiona a 10 que es suficiente para esta prueba y la mayoría de cálculos. La dimensión de esta pila indica el número máximo de paréntesis anidados que permitimos en las operaciones. Es decir, se permiten hasta 10 paréntesis anidados. ¡Realmente son expresiones muy poco usuales!

La segunda parte a diseñar es el bucle de órdenes.

Un esquema puede ser el siguiente:

```
100 REM Inicio del bucle de órdenes
110 GO SUB 9000 : REM Escribir el estado de la memoria
120     INPUT "Orden:";o$
199     REM Inicio del bloque condicional para el reparto
200          IF o$ <> "+" THEN GO TO 300
210                 GO SUB 6500 : REM Operación
220                 GO TO 900
300          IF o$ <> "-" THEN GO TO 400
310                 GO SUB 6500
320                 GO TO 900
400          IF o$ <> "*" THEN GO TO 500
410                 GO SUB 6500
420                 GO TO 900
500          IF o$ <> "/" THEN GO TO 600
510                 GO SUB 6500
520                 GO TO 900
600          REM Parte de otro caso (se debe memorizar)
610          INPUT "Número:";n
620          GO SUB 6000 : REM Memorizar
900     REM Fin del bloque condicional
990     GO TO 100
```

Se distinguen dos partes:

a) El bloque condicional que va desde la línea 200 hasta la 900. Realiza el papel de distribuir las cosas a hacer según la orden tecleada.

b) El bloque que constituye el bucle de órdenes que se inicia en la línea 100, llama a una subrutina y la instrucción INPUT, sigue a continuación el bloque condicional y finalmente se vuelve a reciclar hasta el inicio.

Observe que realizamos un bucle sin terminación. De momento vamos a dejarlo así. Más adelante definiremos cómo hay que acabar. En estos momentos habrá que acabar mediante la tecla de STOP en la entrada de la orden.

En el bloque condicional analizamos cinco casos, cuando el símbolo entrado es un signo aritmético (+, –, * y /) se envía a la subrutina 6500 que realiza la operación.

En caso contrario, se supone que desea memorizar algún número y lo pregunta en la línea 610, una vez entrado se envía el control a la subrutina 6000 que lo memoriza sobre la memoria de pila. La variable *n* contiene el número entrado.

Por otra parte, la subrutina 9000 se encarga de visualizar el estado de la memoria de pila.

Para terminar este prototipo es necesario programar las subrutinas 6000, 6500 y 9000. Empecemos por la 9000, el texto es el siguiente:

```
9000 REM Subrutina para visualizar la memoria.
9010 FOR i = 1 TO pp
9020 PRINT i,p(i)
9030 NEXT i
9040 PRINT "-----------------------------"
9050 RETURN
```

Simplemente visualiza en una línea el número de orden de la memoria y su contenido mediante un bucle FOR/NEXT. Una vez terminado imprime unos guiones para indicar la separación entre un caso y el siguiente.

La subrutina 6000 es también fácil de realizar. En efecto, simplemente hay que incrementar el orden *pp* y colocar en *p(pp)* el valor de *n*, que es el valor entrado.

El esquema es el siguiente:

```
6000 REM Subrutina para memorizar
6010 LET pp = pp +1 : LET p(pp) = n
6020 RETURN
```

En la rutina 6500 hay que hacer la operación. Esta rutina es algo más complicada, pero se debe al hecho de que hay que hacer cuatro operaciones distintas. El programa es el siguiente:

```
6500 REM Subrutina para operar
6510 LET pp = pp -1
6520 REM bloque condicional
```

```
6530      IF o$ <> "+" THEN GO TO 6600
6540         LET p(pp) = p(pp) + p(pp+1)
6550         GO TO 6900
6600      IF o$ <> "-" THEN GO TO 6700
6610         LET p(pp) = p(pp) - p(pp+1)
6620         GO TO 6900
6700      IF o$ <> "*" THEN GO TO 6800
6710         LET p(pp) = p(pp) * p(pp+1)
6720         GO TO 6900
6800      IF o$ <> "/" THEN GO TO 6900
6810         LET p(pp) = p(pp) / p(pp+1)
6820         GO TO 6900
6900 RETURN
```

El mecanismo es en primer lugar rebajar en 1 el indicador de hasta dónde esta llena la pila.

A continuación se entra en un bloque condicional que selecciona la operación a realizar.

No existe bloque de otro caso porque se supone que cuando se llega a esta subrutina sólo se llega para valores permitidos de la operación. Por lo tanto, el IF de la línea 6800 y el GO TO de la línea 6820 son inútiles. Los dejamos así para añadir operadores, si conviene, con facilidad.

Observe que se utiliza para operar el elemento *pp + 1*. Se hace así porque estamos seguros de que, cuando hemos entrado en la subrutina, este elemento contiene un valor correcto. De hecho la primera operación que se ha realizado es decrementar en 1 el indicador del último ocupado de la memoria.

Finalmente las operaciones sumar y multiplicar son conmutativas; es decir, da igual sumar 3 más 4 que 4 más 3. No lo son, en cambio, las operaciones de restar y dividir. Vea que se supone que se resta el más reciente entrado del penúltimo, es decir del que está en posición anterior en la pila. El mismo razonamiento se aplica para la división.

El programa se muestra completo en la figura 1.

13.2.3 Pruebas para el prototipo

Experimentemos un poco con esta máquina de calcular que hemos construido de una manera muy simple.

Experimento 1. Calcular la expresión que se ha puesto como ejemplo, es decir,

$$3 * (4 + 2).$$

Ejecute el programa, le aparece en pantalla una línea de guiones porque la memoria esta vacía. A continuación le aparece la petición de una orden. Teclee el ENTER para indicar que quiere memorizar. Le pide a continuación el número, teclee un 3 y un ENTER.

Figura 1: Programa del prototipo de calculadora

```
  10 REM -- Inicializaciones
  20 DIM p(10)
  30 LET pp = 0

 100 REM Inicio del bucle de órdenes
 110 GO SUB 9000 : REM Escribir el estado de la memoria
 120     INPUT "Orden:";o$
 199     REM Inicio del bloque condicional para el reparto
 200         IF o$ <> "+" THEN GO TO 300
 210               GO SUB 6500 : REM Operación
 220               GO TO 900
 300         IF o$ <> "-" THEN GO TO 400
 310               GO SUB 6500
 320               GO TO 900
 400         IF o$ <> "*" THEN GO TO 500
 410               GO SUB 6500
 420               GO TO 900
 500         IF o$ <> "/" THEN GO TO 600
 510               GO SUB 6500
 520               GO TO 900
 600         REM Parte de otro caso (se debe memorizar)
 610         INPUT "Número:";n
 620         GO SUB 6000 : REM Memorizar
 900     REM Fin del bloque condicional
 990     GO TO 100

6000 REM Subrutina para memorizar
6010 LET pp = pp +1 : LET p(pp) = n
6020 RETURN
6500 REM Subrutina para operar
6510 LET pp = pp -1
6520 REM bloque condicional
6530      IF o$ <> "+" THEN GO TO 6600
6540         LET p(pp) = p(pp) + p(pp+1)
6550         GO TO 6900
6600      IF o$ <> "-" THEN GO TO 6700
6610         LET p(pp) = p(pp) - p(pp+1)
6620         GO TO 6900
6700      IF o$ <> "*" THEN GO TO 6800
6710         LET p(pp) = p(pp) * p(pp+1)
6720         GO TO 6900
6800      IF o$ <> "/" THEN GO TO 6900
6810         LET p(pp) = p(pp) / p(pp+1)
6820         GO TO 6900
6900 RETURN

9000 REM Subrutina para visualizar la memoria.
9010 FOR i = 1 TO pp
9020 PRINT i,p(i)
9030 NEXT i
9040 PRINT "--------------------"
9050 RETURN
```

Le aparece en pantalla un 1 con un 3 en la columna 16 del primer tabulador y una línea de guiones. Esto nos indica que el 3 se ha memorizado en la memoria número 1.

Le pide una nueva orden, se debe memorizar el 4, según la secuencia que se ha escrito antes. Se pulsa un ENTER, para indicar memorización y a continuación el programa nos pide un número. Se le introduce un 4.

En pantalla aparecen las dos primeras memorias que están ocupadas con el contenido de un 3 la primera y un 4 la segunda y la línea de guiones.

Pide a continuación una nueva orden, ahora hay que memorizar el número 2. Se pulsa ENTER para indicar que deseamos memorizar y al pedirnos un número entramos el 2.

Aparecen a continuación en la pantalla 3 líneas que indican que el contenido de las tres primeras memorias, que es 3, 4 y 2 para la primera, la segunda y la tercera.

Ahora toca realizar operación, la de sumar; pulsamos el símbolo + que es el adecuado para indicar la suma y pulsamos ENTER. Ahora el programa realiza la operación y nos presenta la memoria en la pantalla.

Se visualizan sólo las dos primeras, la tercera ha quedado desechada por la realización de la operación, que contienen un 3 y un 6, que proviene de la suma de 4 y 2.

El programa pregunta una nueva orden, hay que realizar la multiplicación, se pulsa el símbolo * y el ENTER y el programa calcula la operación presentando en pantalla una sola línea de memoria con un 18 que proviene de multiplicar 6 por 3.

El resultado de la operación queda en la línea 1 de pantalla.

Si compara todas las operaciones que hemos realizado con los pasos reseñados en el apartado de definición del programa verá que son los mismos.

Experimento 2. Calcular la expresión

5 + 12 * 6.

Esta expresión es parecida a la anterior pero presenta la novedad de que no hay paréntesis y de hecho los necesita. Recuerde que por la prioridad de los operadores esta expresión es equivalente a

5 + (12 * 6)

Por lo tanto, para calcular el resultado es necesario aplicar la secuencia

5 <– – 12 <– – 6 <– – * +

en donde el símbolo <– – significa memorizar.

En nuestro programa, esta secuencia significa:

a) Apretar un ENTER, o cualquier otra tecla excepto las de las operaciones aritméticas. Recuerde que hemos definido que cualquier otro símbolo sirve para indicar que se desea memorizar.

b) Teclear un 5. Aparecen en pantalla dos líneas de memoria con un 18, resultado de los cálculos anteriores, y un 5.

c) Apretar un ENTER. Para memorizar.

d) Teclear un 12. Aparecen tres líneas en pantalla con un 18, un 5 y un 12.

e) Apretar un ENTER. Para memorizar.

f) Teclear un 6. Aparecen cuatro líneas en pantalla con un 18, un 5, un 12 y un 6.

g) Ahora hay que realizar la mutiplicación; para ello introduzca el símbolo * y apriete el ENTER. A continuación aparecen en pantalla tres líneas de memoria, la primera con un 18, la segunda con un 5, la tercera con un 72, que corresponde al producto de 6 por 12.

h) Nos queda por hacer la última suma, basta teclear el símbolo + y apretar un ENTER para obtener dos líneas, la primera con un 18 y la segunda con el resultado de 77, que es la suma de 5 y 72, las dos últimas líneas que había memorizadas antes de teclear la operación.

El resultado 77 queda almacenado en la segunda línea y el 18 continúa en la primera, que es el resultado del cálculo anterior.

Observe que si ahora se desea sumar el resultado del cálculo anterior con el obtenido en este cálculo basta apretar en la orden un símbolo + e inmediatamente las líneas de memoria se han reducido en una unidad y aparece el resultado de 95, que es la suma de 77 y 18.

Experimento 3. Vamos a comprobar que el resultado de restar 3 de 5 depende del orden en que entremos los números.

La situación actual es que disponemos de una línea de memoria con un 95.

Efectuamos primero el cálculo de 5−3. Para ello realizamos los pasos siguientes:

a) Memorizar el 5. Se aprieta el ENTER y se teclea el 5 cuando se nos pide un número. En la pantalla aparecen dos líneas de memoria, la primera con un 95 y la segunda con un 5.

b) Memorizar el 3. Se aprieta el ENTER y se teclea el 3 cuando se nos pide el número. En la pantalla aparecen tres líneas de memoria que contienen un 95, un 5 y un 3.

c) Se teclea el signo −, para indicar la resta de las dos últimas memorias. En pantalla aparecen ahora dos líneas, una con un 95, de resultados anteriores, y la segunda con un 2, que corresponde a la resta de 5 menos 3.

Comparemos este resultado con la operación inversa, es decir, 3 menos 5.

Los pasos son los siguientes:

a) Memorizar el 3. La pantalla queda como

```
1    95
2    2
3    3
```

b) Memorizar el 5. La pantalla queda como

1	95
2	2
3	3
4	5

c) Entrar el signo menos. La pantalla queda como

1	95
2	2
3	−2

Observe cómo el resultado es de signo contrario al obtenido en el caso anterior.

Aplique el + dos veces pulsando ENTER entre cada signo para obtener la siguiente pantalla

1	95

Experimento 4. Si en la situación que estamos ahora pulsa una operación aritmética, por ejemplo, la suma, ¿qué ocurre? Pues bien, como sólo tenemos una línea el programa debería decir que no se puede hacer la operación. Como no hemos realizado ningún control de que la operación se pueda realizar el programa da el error

3 Subscript wrong, 6540:1

ya que se intenta encontrar el elemento cero de la tabla, que no existe.

Experimento 5. Vuelva a teclear el RUN del programa y empiece a memorizar números. Siga la secuencia de 1, 2, 3, 4, ... hasta 10, en el momento de introducir el 11 el programa le dará el error de

3 Subscript wrong, 6010:2

porque se ha sobrepasado la memoria disponible que hemos definido 10 para el tamaño.

Experimento 6. Vuelva a teclear el RUN. Memorice un número cualquiera, por ejemplo, el 3. A continuación memorice el 0. Finalmente intente hacer la división; como resulta que el divisor es cero aparece un error de

6 Number too big, 6810:1

Las conclusiones de la manipulación de este prototipo son las siguientes:

a) Se ha aprendido a manipular las expresiones con el método que indica esta calculadora.

b) Es necesario establecer el control cuando se quiere hacer una operación y sólo tenemos una memoria disponible.

c) Es necesario establecer un control cuando se sobrepasa el nivel de memorias que hemos destinado.

d) Se ha de controlar que no se realicen divisiones por cero.

13.2.4 Diseño completo

En las pruebas que hemos realizado con el prototipo se han detectado una serie de defectos que eliminar en el diseño completo que ahora empezamos.

Por otra parte, no se preocupe demasiado de la correspondencia de esta calculadora con las expresiones que normalmente utilizamos en el BASIC. Existen unas reglas que permiten pasar de las expresiones con paréntesis a las que precisa esta calculadora. Sin embargo, éste no es el objetivo de esta práctica. Unicamente se ha utilizado para aprender el manejo de las reglas de esta calculadora que son un poco distintas de las calculadoras habituales.

Para clarificar este punto, las reglas que rigen esta calculadora son las siguientes:

a) Dispone de una memoria en forma de pila.

b) La operación memorizar consiste en introducir un número en la primera memoria desocupada.

c) Las operaciones aritméticas toman los dos últimos elementos de la pila ocupados, realizan la operación con ellos y colocan el resultado en la memoria que lleva el orden más bajo.

d) Después de una operación aritmética, la memoria que queda libre es la que contiene el segundo operando. Es decir, la que tiene el número de orden más alto.

e) De hecho las reglas *c* y *d* deben considerarse para operadores binarios como lo son los aritméticos de sumar, restar, multiplicar y dividir.

Si se quieren introducir operadores unarios, es decir, que sólo modifican un valor, hay que añadir la regla siguiente:

e) Los operadores unarios ejecutan la función sobre el elemento más alto de la pila y el resultado lo depositan en esta memoria.

Los operadores unarios no suben ni bajan la pila (nos referimos que operan sobre el último elemento de la memoria sin alterar el grado de ocupación de la memoria).

Los operadores unarios más frecuentes en una calculadora son, cambiar de signo, tomar el valor absoluto, sacar la parte entera, sacar la parte fraccionaria (es lo mismo que tomar la parte entera), etc.

En el enunciado del problema no se ha hablado de operadores unarios. Sin embargo, en el diseño completo vamos a considerarlos ya que es ne-

cesario disponer de números negativos y la mejor manera es utilizar el operador de cambio de signo.

Una vez considerado un operador unario los demás son relativamente fáciles de añadir.

Del mismo modo, es posible pensar en otros operadores binarios tales como sacar el módulo. No obstante, de momento trabajaremos con los que se han utilizado hasta ahora.

Por otra parte, las teclas de órdenes se van a limitar a las siguientes:

Operadores binarios

+ Suma

− Resta

* Multiplicación

/ División

Operadores unarios

<> Cambio de signo.

Observe que hay que elegir un símbolo distinto para la resta y el cambio de signo, pues la máquina ha de distinguir entre un proceso de cambio de signo que es unario y un proceso de resta que es binario.

Operaciones adicionales

ENTER	Memorización de un número.
c	Volver a la posición inicial de memoria.
f	Finalizar el programa.

Cualquier pulsación de otra tecla se considera errónea. Por otra parte, las teclas que son letras deben ignorarse si son en mayúsculas o minúsculas.

La tecla *f* cubre la necesidad de finalizar el programa adecuadamente.

La tecla *c* es la que permite volver toda la memoria a la condición inicial. Es como si finalizamos y volvemos a ejectuar el programa.

Los problemas a resolver ahora son de tres tipos:

a) Conseguir que la máquina de calcular no dé errores. Es decir, hay que establecer el control de la memoria.

b) Mejorar, en lo posible, la presentación de resultados.

c) Mejorar, en lo posible, la entrada de datos.

El primer problema es el más inmediato a resolver.

13.2.4.1 El control de la memoria

Este control debe ejercerse en dos puntos, cuando realizamos las operaciones binarias y tenemos uno o ningún elemento en memoria y cuando intentamos memorizar un elemento y la memoria está llena.

Desde el punto de vista del programa prototipo estos controles deben colocarse en la subrutina 6000, para evitar el desbordamiento por exceso de memorias ocupadas antes de la línea 6010, que es donde se incrementa el valor del indicador de la última memoria ocupada (*pp*).

También hay que colocarlos antes de la línea 6510 para evitar que se realice la operación cuando no es posible.

Por otra parte deberemos añadir una subrutina para el tratamiento de operadores unarios, que debe contemplar el control de memoria únicamente cuando empecemos, es decir, cuando no disponemos de ningún valor en memoria.

El programa de estas subrutinas queda como:

```
      6000 REM Subrutina para memorizar
--->  6010 IF pp < ls THEN GO TO 6050
--->  6020    LET e=1 : GO SUB 9100
--->  6030    GO TO 6060
      6050    LET pp = pp +1 : LET p(pp) = n
      6060 RETURN

--->  6100 REM Subrutina para los operadores unarios.
--->  6110 IF pp > 0    THEN GO TO 6150
--->  6120     LET e=2 : GO SUB 9100
--->  6130     GO TO 6199
--->  6140     REM bloque condicional de operadores unarios
--->  6150     IF o$ <> "<>" THEN GO TO 6160
--->  6151        LET p(pp) = - p(pp) : GO TO 6199
--->  6199 RETURN

      6500 REM Subrutina para operar
--->  6510 IF pp >1 THEN GO TO 6550
--->  6520    LET e=3 : GO SUB 9100
--->  6530    GO TO 6900
      6550    LET pp = pp -1
      6560    REM bloque condicional
      6570        IF o$ <> "+" THEN GO TO 6600
      6580           LET p(pp) = p(pp) + p(pp+1)
      6590           GO TO 6900
      6600        IF o$ <> "-" THEN GO TO 6700
      6610           LET p(pp) = p(pp) - p(pp+1)
      6620           GO TO 6900
      6700        IF o$ <> "*" THEN GO TO 6800
      6710           LET p(pp) = p(pp) * p(pp+1)
      6720           GO TO 6900
      6800        IF o$ <> "/" THEN GO TO 6900
--->  6810           IF p(pp+1) <> 0 THEN GO TO 6850
--->  6820              LET e=4 : GO SUB 9100
--->  6830              GO TO 6900
      6850              LET p(pp) = p(pp) / p(pp+1)
      6860              GO TO 6900
      6900 RETURN
```

Las líneas añadidas respecto al programa anterior se marcan con una flecha al lado.

Las consideraciones más importantes a tener en cuenta son:

— En la subrutina de memorizar se ha incluido un bloque condicional simple (línea 6010 hasta línea 6060) que impide la memorización si ya es-

tamos en la última memoria de la pila. Se llama a la subrutina 9100 con el valor de la varable *e* igual a 1. Esta subrutina es la encargada de visualizar los errores.

— También en esta subrutina se utiliza el valor de *ls* para indicar cuál es el valor máximo de la memoria que disponemos. Esta constante se debe definir como las constantes que tiene el programa en la primera fase de inicializaciones. Evitamos de esta manera sepultar un número en un programa con el inconveniente de que si hay que modificarlo se tiene que hacer en muchos lugares. De esta manera se puede cambiar el tamaño de la memoria solamente cambiando una instrucción.

— En la subrutina de los operadores binarios se añade el control inicial de que si la memoria última ocupada es la primera no se puede realizar la operación. Se coloca el valor de *e* a tres y se envía a la subrutina de visualización del error.

Se estructura mediante un bloque condicional de alternativa simple que se inicia en la línea 6510. La parte ENTONCES comprende desde la línea 6520 hasta la línea 6530. La parte EN CASO CONTRARIO comprende desde la línea 6550 hasta la 6860.

— Después se coloca en la parte correspondiente a la división el control de la división por cero. Es un bloque de alternativa simple que comprende desde la línea 6810 hasta la 6860. Observe que si se da el caso de esta división por cero, realmente se rebaja la memoria de la pila en uno, es decir, se rechaza el cero, y se deja el valor que había sin cambiar. En el caso de un error es difícil seguir la política más conveniente en cada caso. Esta que se elige es la que perturba menos el programa y la calculadora.

— La subrutina que se ha añadido controla en primer lugar si estamos al inicio, donde la pila no contiene ningún elemento. Si esto es cierto, se da un error y no se hace absolutamente nada. Las líneas que controlan esta decisión son la 6110 y la 6130. Si no se dan las condiciones de error la subrutina entra en un bloque condicional para analizar de qué operador unario se trata. Se da esta estructura de bloque condicional de alternativa múltiple pues, aunque sólo tenemos un operador, en el futuro pueden añadirse más y en este caso es conveniente tener preparado el programa para esta adición (líneas 6150 hasta 6151).

— El retoque de esta subrutina ha generado un mecanismo de tratamiento de los errores que hay que contemplar en el aspecto de la presentación de resultados.

13.2.4.2 La presentación de resultados

En primer lugar, la presentación de resultados de la máquina del prototipo es útil porque describe la evolución de la máquina de calcular. Una vez visto esto es más conveniente pensar en una presentación de resultados más agradable.

Es interesante pensar en que las memorias de la máquina se inicien en el borde superior de la pantalla con la memoria 1, incluso con la posi-

bilidad de un rótulo orientativo. A medida que memorizamos las líneas se presentan en pantalla creciendo y decreciendo según las operaciones realizadas.

De esta manera los errores se pueden visualizar en una línea hacia el final de la pantalla.

Además, el hecho de que se produzcan errores obliga a tener una acción para borrarlos cuando la condición ha desaparecido. Para centralizar el tratamiento de los errores es conveniente imprimir el error cero como el caso normal, es decir, sin error. De esta manera el borrar un error es lo mismo que escribir el mensaje de sin error.

Para ello crearemos una tabla de errores de tal manera que el tratamiento sea uniforme.

Las rutinas de visualización de memoria y errores quedan de la manera siguiente:

```
    9000 REM Subrutina para visualizar la memoria.
--> 9010 PRINT AT 1,1;
--> 9020 PRINT "MEMORIA NO.","VALOR"
--> 9030 PRINT "-------------------------------"
    9040 FOR i = 1 TO pp
    9050 PRINT i,p(i)
    9060 NEXT i
    9070 PRINT "                               "
    9080 RETURN

--> 9100 REM Subrutina para visualizar errores
--> 9110 PRINT AT 20,1;m$(e+1);
--> 9120 IF e<>0 THEN BEEP 0.25,1
--> 9130 RETURN
```

Como antes, las líneas añadidas o modificadas se marcan con una flecha.

Respecto a las visualización de la memoria se han añadido las líneas 9010, 9020 y 9030. La primera se coloca al inicio de la pantalla, las dos siguientes colocan un rótulo.

Se visualizan entonces todas las líneas de memoria. A continuación se escribe una línea en blanco. Si se ha realizado una operación, nos borra la línea siguiente que estaba ocupada. De esta manera todas se visualizan siempre desde el borde superior de la pantalla.

Las líneas de guiones y de blancos contienen 31 blancos para conseguir eliminar toda una línea de pantalla.

Respecto a la subrutina de tratamiento de los errores, se inicia con la impresión en la columna 1 de la fila 20 del mensaje correspondiente.

La línea siguiente pregunta si el error es distinto de cero. En el caso afirmativo, se hace un sonido para advertir al usuario de la existencia de error.

No se hace en el caso de que el error sea cero, pues hemos dicho que se trataba de borrar el error, con lo que no se ha de advertir al usuario.

Observe que la instrucción PRINT acaba en punto y coma, pues ya sabemos seguro que no nos interesa ir a la línea siguiente.

El mensaje de error que se escribe es el que corresponde al elemento de la tabla más 1; se debe a que el ZX-Spectrum no tiene el índice cero de las tablas y, por lo tanto, hay que guardar los mensajes desplazados una unidad.

Finalmente para tener todo completo respecto a estas mejoras es necesario inicializar las variables.

Cambiamos las inicializaciones del programa a:

```
10 REM Inicializaciones de constantes.
11 LET ls = 10 :REM Tamaño de la memoria de la pila.
20 DIM m$(6,30):REM Mensajes de error
21 LET m$(1) = ""
22 LET m$(2) = "Se ha terminado la memoria"
23 LET m$(3) = "No hay ningún elemento"
24 LET m$(4) = "No se puede operar"
25 LET m$(5) = "División por cero"
26 LET m$(6) = "Tecla incorrecta"
50 DIM p (ls) :REM Dimensionado de la memoria.
90 REM Inicialización de las variables
91 LET pp= 0 : REM Memoria vacía.
```

En la tabla *m$* se colocan los diferentes mensajes de error que pueden aparecer. En el ZX-Spectrum es necesario dar el tamaño de cada una de las cadenas de caracteres que constituye la tabla. En este caso se ha tomado 30.

Como el ZX-Spectrum rellena las longitudes de *m$* hasta 30 caracteres con blancos, el elemento primero de la tabla sirve también para escribir una línea en blanco. Así la línea 9070 de la subrutina de visualizar la memoria queda como

```
9070 PRINT m$(1)
```

También se define el tamaño máximo de la memoria asignando a *ls* el valor del 10.

13.2.4.3 El bucle de órdenes

La definición rigurosa de la máquina de calcular nos hace variar el bucle de órdenes.

La razón principal para cambiar el bucle de órdenes es que hay que clasificar la orden según sea un operador binario, un operador unario, un comando especial o la memorización. Además se debe incluir que cualquier otra tecla pulsada que no sea alguna de las anteriores debe dar señal de error.

Para agilizar la entrada es conveniente que en lugar de utilizar un INPUT para la orden se utilice la tecla INKEY$, ya que evita el tener que pulsar un ENTER después de cada orden.

Una vez obtenido el carácter correspondiente es necesario clasificarlo. Para ello se utiliza una variable *c* que sólo toma los valores de 12 si se trata de memorizar, de 1 si se trata de un operador unario, de 2 si se trata de un operador binario, de 10 si se trata de finalizar, de 11 si se trata de volver a la situación inicial. Cuando el símbolo no está permitido se coloca un cero.

Para organizar esta clasificación de modo que sea fácil incorporar nuevas funciones se realiza del modo siguiente:

El número de teclas que se pueden dar son de 255, representan los valores de 1 a 255 del código ASCII que da el teclado. Se dispone una tabla en memoria de 255 valores que contienen el término de clasificación de cada uno de los caracteres.

Prácticamente el esquema de la rutina es el siguiente:

```
1000 REM Tabla de clasificación de símbolos
1110 DIM c(255)
1120 LET c(13) = 12 : REM símbolo de memorización
1130 LET c ( CODE ( "f") ) = 10
1140 LET c ( CODE ( "F") ) = 10
1150 LET c ( CODE ( "+") ) = 2
1160 LET c ( CODE ( "-") ) = 2
1170 LET c ( CODE ( "*") ) = 2
1180 LET c ( CODE ( "/") ) = 2
1190 LET c ( CODE ( "<>" )) = 1
1200 LET c ( CODE ( "c" ) ) = 11
1210 LET c ( CODE ( "C" ) ) = 11
1220 RETURN
```

Esta subrutina sólo se realiza una vez al inicio del programa para colocar los valores de la clasificación correctamente.

La tabla *c* es de 255 valores, el subíndice de esta tabla está asociado al código ASCII de un carácter. Así en el código ASCII el espacio en blanco es el código 32, quiere decir esto que *c(32)* me debe dar los atributos de clasificación de esta tecla.

En principio cuando se genera esta tabla todos sus elementos se colocan a cero. Consideraremos este clasificador como indicación de error.

Inicializamos el resto de códigos útiles a sus valores. Se inicia con la situación en el valor 13, que corresponde a la tecla ENTER, con el valor de 12, que es el valor reservado para memorizar.

A continuación se coloca un 10 en las casillas de las letras f y F que indican finalización.

La técnica consiste en utilizar la función CODE que devuelve el código ASCII del primer carácter de la cadena. Este resultado sirve de subíndice para la tabla *c*. De esta manera queda mucho más claro a qué valor nos referimos. Además sólo se pasa una vez por esta subrutina y la eficiencia no debe importarnos mucho.

¿Se preguntará por qué esta complicación? La razón es que cuando demos a una tecla no hará falta recorrer una tabla. Mediante el código ASCII de la tecla entrada obtendremos las propiedades de clasificación. El tener que buscar en una tabla es un método mucho más largo en el tiempo que utilizar un subíndice.

Se definen a continuación con el mismo método los operadores binarios (el valor de *c* se coloca a 2) +, –, * y /. Los operadores unarios (el valor es 1) <> y la función de iniciar que se le coloca al valor 11.

La subrutina de entrar una orden queda como la siguiente:

```
1500 REM Subrutina de entrada de órdenes
1510 PAUSE 0 : LET o$=INKEY$
1520 LET e=0 : GO SUB 9100
1530 LET cc = c ( CODE(o$))
1540 IF cc <>0 THEN RETURN
1550     LET e=5 : GO SUB 9100
1560     GO TO 1500
```

El bloque se describe como un bucle que sólo se abandona en el caso de que la tecla que se ha obtenido sea la correcta. Esto se hace en el IF de la línea 1540, cuando el elemento de la tabla que se ha construido sea un valor clasificado.

La línea 1510 lee un carácter del teclado. La línea 1520 elimina los errores visualizados en la línea de errores.

La lína 1530 obtiene en *cc* el valor de clasificación de la tecla pulsada. En la línea siguiente se analiza si este valor no es erróneo. En caso afirmativo se ha obtenido la orden y su clasificación y se devuelve el control al programa o subrutina que ha llamado.

En caso de que el valor sea erróneo se envía un mensaje de error y se espera una orden correcta.

La figura 2 contiene el programa completado con estas cuestiones. Repáselo para ver si está completo y tenga en cuenta estas cuestiones.

Los detalles que hay que observar son:

— Se ha añadido la línea 92 que envía el programa a la subrutina 1000, que es la que corresponde a la inicialización de la matriz de clasificación.

— En lugar de hacer el INPUT para recoger la orden se sustituye por la subrutina de entrada diseñada con el INKEY$ y el tratamiento de error. Observe el cambio de la línea 120.

— Inmediatamente después de la vuelta de la subrutina de órdenes (línea 130) se pregunta si se ha dado la orden de finalizar; en caso afirmativo se finaliza el programa mediante el envío a la línea 9999. Con esta instrucción se ha transformado el bucle sin salida que comprendía desde la línea 100 hasta la 990, en un bucle generalizado cuya salida se realiza en la pregunta de la línea 130.

— El bucle de órdenes ha variado la variable sobre la que se apoya la clasificación. Antes se hacía según la orden llegada; ahora se hace mediante la variable de clasificación.

— Esta estructura condicional tiene cuatro ramas, cuando *cc* vale 1 hay que ir a la subrutina de operadores unarios que es la 6100, cuando vale

Figura 2: Programa de simulación de una calculadora con memoria de pila

```
  10 REM Inicializaciones de constantes.
  11 LET ls = 10 :REM Tamaño de la memoria de la pila.
  20 DIM m$(6,30):REM Mensajes de error
  21 LET m$(1) = ""
  22 LET m$(2) = "Se ha terminado la memoria"
  23 LET m$(3) = "Nc hay ningún elemento"
  24 LET m$(4) = "No se puede operar"
  25 LET m$(5) = "División por cero"
  26 LET m$(6) = "Tecla incorrecta"
  50 DIM p(ls) : REM Dimensionado de la memoria.
  90 REM Inicialización de las variables
  91 LET pp = 0 : REM Memoria vacía.
  92 GO SUB 1000: REM Clasificador

 100 REM Inicio del bucle de órdenes
 110 GO SUB 9000 : REM Escribir el estado de la memoria
 120    GO SUB 1500 : REM Obtener la orden
 130    IF cc = 10 THEN GO TO 9999
 199    REM Inicio del bloque condicional para el reparto
 200        IF cc <> 1 THEN GO TO 300
 210              GO SUB 6100 : REM Operación unaria
 220              GO TO 900
 300        IF cc <> 2 THEN GO TO 400
 310              GO SUB 6500 : REM Operación binaria
 320              GO TO 900
 400        IF cc <> 11 THEN GO TO 600
 410              GO SUB 9200  : REM Iniciar otra vez
 420              GO TO 900
 600        REM Parte de otro caso (se debe memorizar)
 610        INPUT "Número:";n
 620        GO SUB 6000 : REM Memorizar
 900    REM Fin del bloque condicional
 990    GO TO 100

1000 REM Tabla de clasificación de símbolos
1110 DIM c(255)
1120 LET c(13) = 12 : REM símbolo de memorización
1130 LET c ( CODE ( "f") ) = 10
1140 LET c ( CODE ( "F") ) = 10
1150 LET c ( CODE ( "+") ) = 2
1160 LET c ( CODE ( "-") ) = 2
1170 LET c ( CODE ( "*") ) = 2
1180 LET c ( CODE ( "/") ) = 2
1190 LET c ( CODE ( "<>" )) = 1
1200 LET c ( CODE ( "c" ) ) = 11
1210 LET c ( CODE ( "C" ) ) = 11
1220 RETURN

1500 REM Subrutina de entrada de órdenes
1510 PAUSE 0 : LET o$=INKEY$
1520 LET e=0 : GO SUB 9100
1530 LET cc = c( CODE(o$) )
1540 IF cc <>0 THEN RETURN
1550    LET e=5 : GO SUB 9100
1560    GO TO 1500


6000 REM Subrutina para memorizar
6010 IF pp < ls THEN GO TO 6050
6020    LET e=1 : GO SUB 9100
```

```
6030      GO TO 6060
6050      LET pp = pp +1 : LET p(pp) = n
6060 RETURN

6100 REM Subrutina para los operadores unarios.
6110 IF pp > 0   THEN GO TO 6150
6120      LET e=2 : GO SUB 9100
6130      GO TO 6199
6140      REM bloque condicional de operadores unarios
6150      IF o$ <> "<>" THEN GO TO 6160
6151         LET p(pp) = - p(pp) : GO TO 6199
6199 RETURN

6500 REM Subrutina para operar
6510 IF pp >1 THEN GO TO 6550
6520    LET e=3 : GO SUB 9100
6530    GO TO 6900
6550    LET pp = pp -1
6560    REM bloque condicional
6570        IF o$ <> "+" THEN GO TO 6600
6580           LET p(pp) = p(pp) + p(pp+1)
6590           GO TO 6900
6600        IF o$ <> "-" THEN GO TO 6700
6610           LET p(pp) = p(pp) - p(pp+1)
6620           GO TO 6900
6700        IF o$ <> "*" THEN GO TO 6800
6710           LET p(pp) = p(pp) * p(pp+1)
6720           GO TO 6900
6800        IF o$ <> "/" THEN GO TO 6900
6810           IF p(pp+1) <> 0 THEN GO TO 6850
6820              LET e=4 : GO SUB 9100
6830              GO TO 6900
6850              LET p(pp) = p(pp) / p(pp+1)
6860              GO TO 6900
6900 RETURN


9000 REM Subrutina para visualizar la memoria.
9010 PRINT AT 1,1;
9020 PRINT "MEMORIA NO.","VALOR"
9030 PRINT "--------------------------------"
9040 FOR i = 1 TO pp
9050 PRINT i,p(i)
9060 NEXT i
9070 PRINT m$(1)
9080 RETURN

9100 REM Subrutina para visualizar errores
9110 PRINT AT 20,1;m$(e+1);
9120 IF e<>0 THEN BEEP 0.25,1
9130 RETURN

9200 REM Subrutina para volver a situación inicial
9210 PRINT AT 3,1;
9220 FOR i= 1 TO pp
9230 PRINT m$(1)
9240 NEXT i
9250 LET pp =0
9260 RETURN
```

2 se envía a la subrutina de operadores binarios la 6500, si se quiere volver a iniciar se envía a la subrutina 9200 que se describe a continuación. En el otro caso se pide un número, se supone que se trata de memorizar. Esto es así porque el error ya se trata en la rutina de entrada.

— Observe que ahora esta estructura condicional nos reparte por categorías de órdenes según el esquema de clasificación. En las rutinas de tratamiento ya existe la segunda estructura condicional para la orden contenida en la variable *o$*.

— La subrutina 9200 nos vuelve a la situación inicial. Básicamente tiene una doble misión: primero, borrar todas las memorias que están en pantalla en este momento, y se consigue mediante el bucle de 9220 hasta 9240. La línea 9210 sitúa el inicio de la impresión debajo de los rótulos, es decir, en la línea 3 de pantalla. La segunda consiste en dejar la memoria vacía, esto se consigue mediante la línea 9250 que indica que la última memoria ocupada es la cero.

Una vez tecleado el programa, experimente con él, haga los mismos experimentos que antes. Sobre todo observe que cuando se produce un error la máquina responde con un mensaje.

Es importante que pruebe también el operador de cambio de signo, observe que en la visualización hay una superposición de números. Por ejemplo, apriete una *c* para volver al inicio. Intente cambiar de signo, la máquina le responde con el error de que la memoria está vacía. Pulse el ENTER y coloque un 57. Cambie de signo, en la pantalla aparece −57. Si ahora vuelve a cambiar en la pantalla aparece 577, de hecho este número es falso, realmente en la memoria hay un 57, lo que ocurre es que como −57 tiene tres cifras y 57 sólo dos el último 7 corresponde a que no se ha borrado la cifra 7.

13.2.4.4 De nuevo la presentación de resultados

Cada vez que presentamos los resultados escribimos la pantalla entera. Esta operación es innecesaria pues sabemos que como máximo se alteran los dos últimos elementos de la memoria. Por otra parte, hay problemas de superposición que se deben evitar, pues si no la calculadora nos da resultados engañosos.

Para ello en lugar de situar la presentación de pantalla al inicio del bucle sólo la vamos a realizar en los momentos oportunos, mejorando además la presentación de los números, ajustando a la derecha y evitando las superposiciones antes mencionadas.

En primer lugar hay que introducir una nueva constante, que es la longitud máxima que deben tener los números cuando se escriben en pantalla.

Añadimos la línea 12 que cumple esta misión y la constante se denomina *ln*, esta línea es:

```
12 LET ln = 14 :REM Longitud máxima de los números.
```

Por otra parte, suprimimos la llamada a la subrutina 9000, desde la línea 110, suprimiendo esta línea. Ahora la presentación debe hacerse a medida que progresa el cálculo. Por ello hay que centrar la atención en las subrutinas 6000, 6100, 6500.

Si se considera la subrutina 6000, es necesario colocar el nuevo valor cuando se ha colocado el número en la memoria. La adición es:

```
6055 LET n$=STR$(n) : GO SUB 9000
```

la nueva subrutina 9000 se encarga de visualizar este nuevo elemento de la memoria, que ya se pasa en forma de cadena de caracteres a la subrutina mediante la variable *n$*. La subrutina 9000 ya sabe dónde debe visualizarlos según el valor de *pp*.

En la rutina 6100, hay que hacer lo mismo a la salida. El nuevo valor calculado por el operador unario se escribe encima del valor antiguo. Las modificaciones que hay que hacer son:

```
6151 LET p(pp) = - p(pp) : GO TO 6198
6198 LET n$=STR$(p(pp)) : GO SUB 9000
```

Observe que hay que tocar la línea 6151 pues el final del bloque condicional de los operadores unarios debe acabar con la visualización de un nuevo valor. En cambio, cuando hay error no hay que visualizar nada.

En las subrutinas de los operadores binarios ocurre que cuando se ha descubierto que no hay error se debe borrar la línea última. Esto se consigue mediante la introducción de

```
6550 LET n$="" : GO SUB 9000
6555 LET pp = pp-1
```

Se envía como argumento la cadena vacía.

Una vez realizada la operación, hay que visualizar el resultado. Esto ocurre a la salida del bloque condicional de alternativa múltiple. Observe otra vez que el final del bloque condicional del error, que es de alternativa simple, y el final del bloque condicional de alternativa múltiple, que corresponde a las operaciones, son los mismos. Hay que hacer los cambios siguientes:

```
6530 GO TO 6910
6900 LET n$=STR$(p(pp)) : GO SUB 9000
6910 RETURN
```

La primera independiza los finales de los dos bloques. La segunda envía a visualizar después del bloque de operación. Y la tercera constituye el nuevo final del bloque de alternativa simple.

La subrutina de visualización corresponde ahora a:

```
9000 REM Rutina de visualización
9010 IF LEN(n$) > ln THEN  LET n$ = n$(1 TO ln-1)+"*"
9020 IF LEN(n$) < ln THEN  LET n$=" "+n$ : GO TO 9020
9030 PRINT AT pp+2,1;n$;
9040 RETURN
```

En la línea 9010 se corta el número si tienen más cifras que las que deseamos en la visualización y se añade un asterisco para indicar que se ha producido el corte. Observe que el número en memoria no se ha alterado, sólo es un efecto de presentación.

En la línea 9020 se ajusta *n$* a la derecha añadiendo blancos hasta la longitud adecuada, marcada por el valor de *ln*.

La línea 9020 imprime en el valor de *n$*. Se ha suprimido en este caso la escritura del número de memoria.

Finalmente, para que la nueva presentación quede completa es preciso que el rótulo se coloque inicialmente. La subrutina 9050 hace esta misión. Esta subrutina es

```
9050 REM Rutina de inicialización de la pantalla
9060 LET n$="VALOR"
9065 IF LEN(n$) < ln THEN  LET n$=" "+n$ : GO TO 9065
9070 PRINT AT 1,1;n$
9075 LET n$=""
9080 IF LEN(n$) < ln THEN  LET n$="-"+n$ : GO TO 9080
9085 PRINT AT 2,1;n$;
9090 RETURN
```

En esta subrutina se sigue la técnica de ajustar los rótulos al valor que indica *ln*. En el caso de que este valor se cambie, los rótulos cambian de tamaño también. En el cálculo de la segunda línea se rellena la variable *n$* con guiones.

Finalmente, antes de empezar hay que llamar a esta subrutina de presentación; por lo tanto, hay que añadir la línea

```
93 GO SUB 9050 : REM Presentación de los rótulos.
```

La figura 3 muestra cómo han quedado los bloques de líneas siguientes: 10–199, 6000–6999 y 9000–9100.

13.2.5 La entrada de datos

El último tema que queda por mejorar es la entrada de datos. Es un poco incómodo tener que teclear el número cada vez que apretamos el ENTER, que a su vez debe acabarse con un ENTER.

Figura 3: Listado de los bloques modificados del programa de una calculadora de memoria de pila

```
  10 REM Inicializaciones de constantes.
  11 LET ls = 10 :REM Tamaño de la memoria de la pila.
  12 LET ln = 14 :REM Longitud máxima de los números.
  20 DIM m$(6,30):REM Mensajes de error
  21 LET m$(1)  = ""
  22 LET m$(2)  = "Se ha terminado la memoria"
  23 LET m$(3)  = "No hay ningún elemento"
  24 LET m$(4)  = "No se puede operar"
  25 LET m$(5)  = "División por cero"
  26 LET m$(6)  = "Tecla incorrecta"
  50 DIM p(ls) : REM Dimensionado de la memoria.
  90 REM Inicialización de las variables
  91 LET pp = 0 : REM Memoria vacía.
  92 GO SUB 1000: REM Clasificador
  93 GO SUB 9050: REM Presentación de rótulos.

 100 REM Inicio del bucle de órdenes
 120     GO SUB 1500 : REM Obtener la orden
 130     IF cc = 10 THEN GO TO 9999
 199     REM Inicio del bloque condicional para el reparto
6000 REM Subrutina para memorizar
6010 IF pp < ls THEN GO TO 6050
6020     LET e=1 : GO SUB 9100
6030     GO TO 6060
6050     LET pp = pp +1 : LET p(pp) = n
6055     LET n$ = STR$(n) : GO SUB 9000
6060 RETURN

6100 REM Subrutina para los operadores unarios.
6110 IF pp > 0   THEN GO TO 6150
6120      LET e=2 : GO SUB 9100
6130      GO TO 6199
6140      REM bloque condicional de operadores unarios
6150      IF o$ <> "<>" THEN GO TO 6160
6151         LET p(pp) = - p(pp) : GO TO 6198
6198      LET n$=STR$ (p(pp)) : GO SUB 9000
6199 RETURN

6500 REM Subrutina para operar
6510 IF pp >1 THEN GO TO 6550
6520     LET e=3 : GO SUB 9100
6530     GO TO 6910
6550     LET n$="" : GOSUB 9000
6555     LET pp = pp -1
6560     REM bloque condicional
6570         IF o$ <> "+" THEN GO TO 6600
6580            LET p(pp) = p(pp) + p(pp+1)
6590            GO TO 6900
6600         IF o$ <> "-" THEN GO TO 6700
6610            LET p(pp) = p(pp) - p(pp+1)
6620            GO TO 6900
6700         IF o$ <> "*" THEN GO TO 6800
6710            LET p(pp) = p(pp) * p(pp+1)
6720            GO TO 6900
6800         IF o$ <> "/" THEN GO TO 6900
6810            IF p(pp+1) <> 0 THEN GO TO 6850
6820               LET e=4 : GO SUB 9100
6830               GO TO 6900
6850               LET p(pp) = p(pp) / p(pp+1)
6860               GO TO 6900
6900     LET n$=STR$(p(pp)) : GO SUB 9000
6910 RETURN
```

```
9000 REM Rutina de visualización
9010 IF LEN(n$) > ln THEN  LET n$ = n$(1 TO ln-1)+"*"
9020 IF LEN(n$) < ln THEN  LET n$=" "+n$ : GO TO 9020
9030 PRINT AT pp+2,1;n$;
9040 RETURN
9050 REM Rutina de inicialización de la pantalla
9060 LET n$="VALOR"
9065 IF LEN(n$) < ln THEN  LET n$=" "+n$ : GO TO 9065
9070 PRINT AT 1,1;n$
9075 LET n$=""
9080 IF LEN(n$) < ln THEN  LET n$="-"+n$ : GO TO 9080
9085 PRINT AT 2,1;n$;
9090 RETURN
```

Una manera más natural de entrar los números es con el mismo mecanismo de INKEY$. Así, se sigue la secuencia que hemos explicado antes, se introduce un número, se memoriza, se opera, etc.

La idea de partida es que, cuando se analiza el carácter de entrada, se permite la entrada de dígitos que se visualizan a medida que los tecleamos, se finaliza la orden mediante una tecla de comando, entendiéndose como tal la que indica si memorizar u operar.

El primer esquema es el siguiente (no lo introduzca).

```
1500 REM Subrutina de entrada de órdenes.
1510 PAUSE 0 : LET o$=INKEY$
1520 LET e=0 : GO SUB 9100
1530 LET cc = c ( CODE(o$) )
1540    IF cc<>0 THEN GO TO 1600
1550       LET e=5 : GO SUB 9100
1560       GO TO 1950
1600    IF no es un dígito THEN GO TO 1700
1610       Construir el número.
1620       GO TO 1950
1700    IF es una orden THEN RETURN
1950 GO TO 1500.
```

Este primer esquema nos indica que hay que añadir en la clasificación los dígitos como un símbolo correcto, que se van a clasificar con un 20. Este número se elige arbitrariamente y no tiene demasiada importancia. También hay que permitir como dígito el punto, ya que se pueden introducir números con decimales. También es cómodo disponer de una tecla que nos permita borrar el último dígito entrado. Esta será la tecla de borrar.

La visión de la entrada de datos de la calculadora ha cambiado, estamos entrando números continuamente y cuando deseamos terminar la entrada le damos una operación que puede ser de dos tipos, o bien memorizar o bien realizar alguna manipulación aritmética.

Centremos la atención en la entrada de los dígitos. Básicamente a medida que se teclea un dígito deben ir apareciendo en pantalla de derecha

a izquierda. ¿Dónde? El lugar más oportuno es en el lugar que corresponde a la primera memoria no ocupada.

Para evitar tener que ajustar a la derecha se trabaja con una cadena de caracteres de longitud fija. Los caracteres se añaden por la derecha y se elimina un blanco por la izquierda. Los caracteres se eliminan mediante la colocación de un blanco a la izquierda y se elimina la cifra de más a la derecha.

Hay que controlar además que sólo aparezca un punto ya que solamente está permitido un punto decimal en los números.

Para desconectar las cuestiones de presentación del número máximo de cifras que se pueden introducir, hay que utilizar otra constante que es la longitud máxima del número. En nuestro caso hemos dejado 14 espacios para la presentación, pero son demasiadas 14 cifras; nuestro ordenador trabaja con menos. Colocamos este valor en 8 cifras.

El esquema de las subrutinas que hacen esta misión es:

```
2000 REM Subrutina de iniciar la entrada de números
2010 LET n$=""
2020 IF LEN(n$) < ln THEN LET n$=" "+n$ : GO TO 2020
2030 LET l= 0
2040 LET punto = 0
2050 RETURN

2100 REM Subrutina de construir el número
2110 IF o$<> r$ THEN GO TO 2200
2120     IF ( punto ) THEN LET punto = n$(ln TO ln) <> "."
2130     LET n$ = " "+ n$(1 TO ln-1)
2140     LET l = l-1
2150     GO TO 2400
2200 IF NOT ( o$="." AND punto) THEN GO TO 2300
2210     LET e = 6 : GO SUB 9100
2220     GO TO 2500
2300 REM llega un carácter que hay que añadir
2310 IF l < lc THEN GO TO 2350
2320     LET e = 7 : GO SUB 9100
2330     GO TO 2500
2350 LET n$ = n$( 2 TO ln ) + o$
2360 LET l = l + 1
2370 IF NOT punto THEN LET punto = o$="."
2400 GO SUB 9000
2500 RETURN
```

La primera subrutina tiene la misión de inicializar los valores necesarios para hacer funcionar la entrada de los números. Se inicializa *n$* a blancos según la longitud que especifica *ln*.

Coloca la variable *l* a cero, esta variable indica el número de dígitos que se han entrado en *n$*; en estos momentos este contador puede parecer innecesario (y a este nivel lo es), pero más adelante se utiliza para diferenciar cuándo se ha entrado número o no.

Finalmente se coloca la variable *punto* a cero, que es una variable lógica, es decir, se coloca a falso. Esta variable sirve para llevar la cuenta de si hay punto decimal o no.

La subrutina 2100 toma diferentes acciones según el dígito específico que llegue. En primer lugar se pregunta si la tecla pulsada corresponde a

r$*. Esta variable *r$ es un valor constante que contiene la tecla que deseamos utilizar para borrar un elemento. Se coloca en forma de variable para agrupar todos estos datos al inicio del programa. En nuestro caso utilizaremos la tecla DELETE, que es la que utiliza el ZX-Spectrum para borrar caracteres. Corresponde al número 12 en código ASCII.

Si corresponde a la tecla de borrar, entonces se alcanza el subbloque de eliminar un carácter por la derecha. Antes de proceder efectivamente a esta eliminación se pregunta si se elimina un punto. De esta tarea se encarga la línea 2120. Si hay punto en la cadena *n$* se coloca la variable *punto* a falso si el último carácter de la cadena es un punto, y a verdadero en caso contrario.

Observe que sólo hay que hacer la pregunta si *punto* es verdadero. Si es falso seguro que no hay un punto; por lo tanto, en este caso el valor de la variable *punto*, si se hiciera la parte THEN del IF, se colocaría a verdadero en contradicción con su significado.

A continuación en la línea 2130 se añade un blanco a la izquierda de *n$* y se fragmenta eliminado el carácter de más a la derecha. Finalmente se descuenta un carácter en la cuenta que lleva *l* y se envía a la línea 2400 que visualiza el valor de *n$* modificado.

El segundo bloque de esta estructura condicional trata el caso de que nos llegue un punto y ya tenemos otro. La pregunta es si *o$* es igual a punto y la variable *punto* es verdadera (línea 2200).

Si la pregunta es cierta se visualiza el error de demasiados puntos.

Finalmente si se alcanza la línea 2300 es que debemos añadir un carácter. Como ya hemos dicho, si se fija un tope máximo que está contenido en la variable *lc*, se pregunta si se ha alcanzado este tope máximo. En caso afirmativo se visualiza el error de demasiadas cifras (línea 2320) y en caso negativo se añade el carácter a *n$* eliminando el carácter de más a la izquierda (línea 2350). Se incrementa la cuenta de *l*. Finalmente si la variable *punto* es falsa se coloca a verdadera, si el carácter que ha llegado es un punto.

Observe otra vez que la pregunta previa es necesaria. Si ya existe un punto no hace falta tocar el valor de punto. Es seguro que el carácter que llega no es un punto y, por lo tanto, debe permanecer en verdadero. Si se ejecuta la parte THEN del IF se coloca en falso cuando llega una cifra distinta de punto.

Finalmente se envía el programa a la subrutina 9000 para que visualice el estado de la cadena *n$* que contiene el número entrado.

Una vez visto el mecanismo de construcción del número ya puede volverse a la construcción de la rutina de entrada de órdenes.

En la figura 4 están las modificaciones que hay que introducir para que funcione correctamente.

En primer lugar se han añadido las líneas 13 y 14 para definir las nuevas constantes, *lc* y *r$*, introducidas.

También se pueden dar dos errores más. Se debe modificar la lína 20 para dimensionar la variable que contiene los errores a 8. Las líneas 27 y 28 añaden los mensajes de error.

Respecto a las líneas 600 hasta 650 las comentaremos más adelante.

Las líneas desde la 1220 hasta la 1270 introducen como símbolos correctos los dígitos decimales, el punto y la tecla de retroceso.

Figura 4: Modificaciones para completar la calculadora de memoria de pila

```
  13 LET lc = 8 : REM Máximo número de cifras
  14 LET r$ = CHR$(12) : REM Carácter de borrado.
  20 DIM m$(8,20) : REM Mensajes de error
  27 LET m$(7)= "Demasiados puntos"
  28 LET m$(8)= "Demasiadas cifras"

 600        REM Parte de otro caso (se debe memorizar)
 610        IF l <> 0 THEN GO TO 900
 620        IF pp < 1 THEN GO TO 640
 630              LET n = p(pp) : GO TO 650
 640              LET n = 0
 650        GO SUB 6000

1220 FOR i = CODE("0") TO CODE("9")
1230 LET c(i) = 20
1240 NEXT i
1250 LET c( CODE(r$) ) = 20
1260 LET c( CODE(".") ) = 20
1270 RETURN

1500 REM Subrutina de entrada de órdenes
1504 LET pp = pp + 1
1508 GO SUB 2000
1510 PAUSE 0 : LET o$=INKEY$
1520 LET e=0 : GO SUB 9100
1530 LET cc = c( CODE(o$) )
1540 IF cc <>0 THEN GO TO 1600
1550    LET e=5 : GO SUB 9100
1560    GO TO 1510
1600 IF cc<> 20 THEN GO TO 1700
1610    GO SUB 2100
1620    GO TO 1510
1700 REM Se ha introducido la orden operar
1710 LET pp = pp -1
1720 IF l<> 0 THEN LET n = VAL(n$) :GO SUB 6000
1730 RETURN

2000 REM Subrutina de iniciar la entrada de números
2010 LET n$=""
2020 IF LEN(n$) < ln THEN LET n$=" "+n$ : GO TO 2020
2030 LET l= 0
2040 LET punto = 0
2050 RETURN

2100 REM Subrutina de construir el número
2110 IF o$<> r$ THEN GO TO 2200
2120    IF ( punto ) THEN LET punto = n$(ln TO ln) <> "."
2130    LET n$ = " "+ n$(1 TO ln-1)
2140    LET l = l-1
2150    GO TO 2400
2200 IF NOT ( o$="." AND punto) THEN GO TO 2300
2210    LET e = 6 : GO SUB 9100
2220    GO TO 2500
2300 REM llega un carácter que hay que añadir
2310 IF l < lc THEN GO TO 2350
2320    LET e = 7 : GO SUB 9100
2330    GO TO 2500
2350 LET n$ = n$( 2 TO ln ) + o$
2360 LET l = l + 1
2370 IF NOT punto THEN LET punto = o$="."
2400 GO SUB 9000
2500 RETURN
```

Los dígitos se hacen mediante un FOR porque en el código ASCII los códigos de los nueve números van seguidos.

La subrutina de entrada empieza en la línea 1500. Se han añadido las líneas 1504 y 1508.

La línea 1504 tiene como misión colocar *pp* como si estuviéramos en la primera memoria desocupada. Se hace con fines únicamente de presentación, para visualizar los números en el lugar correcto. En la línea 1710 hay la instrucción complementaria de disminuir *pp* en 1, que nos devuelven a la situación original cuando se ha terminado la entrada de datos.

La línea 1508 envía el programa a inicializar la entrada de los números según se ha descrito más arriba.

Desde la línea 1510 hasta la 1560 son las mismas instrucciones que antes, excepto la línea 1540, que si la tecla es correcta envía para su análisis a la línea 1600 en lugar de retornar. Por otra parte, la línea 1560 envía el programa a la línea 1510, en lugar de la 1500, ya que no deben volverse a hacer las inicializaciones de las líneas 1504 y 1508.

El bloque 1600-1620 se toma si el carácter entrado es un dígito, un punto o una tecla de retroceso y se envía a la subrutina 2100 que ya hemos descrito. Allí se manipula *n$* y *l* según la instrucción recibida.

Se acaba el bloque con un envío a la línea 1510 para esperar otro carácter.

El bloque 1700-1730 se alcanza cuando hemos recibido un verdadero comando, es decir, o la tecla de memorización o las teclas correspondientes a las operaciones aritméticas. En primer lugar, se decrementa *pp*, para dejar las cosas como estaban antes de entrar en la subrutina. Ya se ha comentado que es por motivos de presentación.

La instrucción 1720 precisa de un comentario más largo. Hasta ahora, se entraban las órdenes puras, en el sentido de que sólo si se quiere memorizar se pide el número. Ahora la política ha cambiado: Se entra un número y para acabar se toca una tecla permitida pero no numérica. De hecho se debe intercalar entre un número y una operación el símbolo de memorizar. Así, si tecleo 33 y a continuación pulso la tecla de la suma +, se debe interpretar que deseo memorizar el 33 y a continuación realizar la suma.

Para controlar este caso se dispone de la variable *l*, si es cero quiere decir que no hemos entrado ningún número y la orden debe interpretarse de una manera pura. En cambio cuando la orden me sirve para finalizar el número debo intercalar la memorización.

Esta es la misión que realiza la línea 1720. Si se ha entrado un número (*l* es distinto de cero) se saca su valor numérico y se memoriza con la función que realiza la subrutina 6000.

Desde la línea 2000 hasta la 2500 ya se han comentado.

Consideremos finalmente el bloque 600 hasta 650. Antes se entraba el número y ahora ya está entrado; en realidad esta parte sobra. Se ha eliminado el INPUT pero hay que considerar otro caso. Hemos hablado de órdenes puras, aquellas en donde no se ha tecleado ningún número. Una de las posibles órdenes es la tecla ENTER, es decir, la memorización de un número. Con la política anterior, cada vez que tecleamos ENTER nos pide un número. Con la política actual ya no es así, es decir, se puede dar el caso de que se toque el ENTER sin haber tecleado un número. Debe considerarse un error. Sin embargo, esta circunstancia puede aprovecharse para hacer una acción que a veces es cómoda, que consiste en duplicar

en la memoria actual lo que hay en la memoria anterior. Aceptando esta nueva regla el bloque se explica fácilmente.

La línea 610 nos envía el control a la línea 900 sin hacer nada. Si se ha entrado un número, esto es debido a que en la línea 1720 ya se ha memorizado debido a la política de hacer una memorización cuando se ha entrado un número.

En cambio, si se ha entrado una memorización pura, es decir, sin número, se debe hacer una operación de memorización con el valor de la última memoria ocupada. Como esto puede ocurrir con la máquina en su situación inicial es necesario prever que se puede encontrar un valor de *pp* igual a cero. Esto también es un error, pero en lugar de señalarlo es mejor suponer que el valor es cero.

Esta idea se consigue con el bloque condicional de alternativa simple que componen las líneas 620, 630 y 640. La línea 650, finalmente, memoriza este valor en la memoria con la llamada a la subrutina 6000.

Pruebe el programa, haga las operaciones de los experimentos mencionados más arriba.

Introduzca también dos pruebas para ver cómo la memorización pura inicialmente coloca un cero en la primera memoria y se continúa en la segunda y mediante la memorización pura repite un valor en la memoria siguiente.

Es posible que en las pruebas si toca las teclas deprisa, y toca más de una a la vez, le aparezca el error de Subscript wrong en la línea 1530. Esto se debe a que en este caso se devuelve el código ASCII cero, que no sirve como subíndice de la tabla de clasificación *c*. Para evitarlo es mejor sustituir la línea 1530 por estas dos.

```
1530  LET cc = CODE (o$)
1535  IF cc <> 0 THEN LET cc = c(cc)
```

Se comprueba si el subíndice es cero antes de llamar a la tabla. Por otra parte, si el subíndice es cero se deja con este valor y se obtiene el mensaje de tecla incorrecta.

13.2.6 Observaciones finales

Hemos realizado una práctica realmente compleja, el interés se centra más en esta complejidad que propiamente en el resultado.

Hemos intentado poner el acento en las cuestiones de diseño, la complicación progresiva del programa ha sido posible debido a que hemos sido cuidadosos en seguir los principios de la programación estructurada.

Incluso hemos desarrollado un prototipo que permite observar con detenimiento las cuestiones fundamentales. Esta manera de trabajar es una de las técnicas más modernas de realización de programas.

Sea consciente de que si vuelve a empezarla encontrará muchos detalles que le han pasado desapercibidos en una primera lectura aunque los programas funcionen. Es difícil captar toda la complejidad de una sola vez.

Por otra parte, es interesante que relacione toda esta manera de operar con la manera de evaluar expresiones por su ordenador. Esencialmente sigue la misma técnica. Además incorpora un algoritmo para traducir sus expresiones en la notación polaca reversa que no hemos dado.

También, así puede comproba mejor el mecanismo de llamada a una subrutina. Cuando llamamos a una subrutina se coloca la dirección de retorno en la primera memoria desocupada de una memoria de pila. En el momento de realizar el RETURN se toma como dirección de retorno, precisamente, la que contiene la última memoria ocupada.

Observe que, en este tipo de mecanismo, la dirección primera en colocarse es la última en salir. Esto permite que una subrutina llame a otra y ésta a otra de una manera natural a base de apilar las direcciones de retorno.

13.3 PRACTICA 2. APROVECHAR EL EDITOR DE TABLA

En el capítulo anterior se ha realizado como práctica un editor de tabla; es decir, un programa que nos permite de una manera cómoda rellenar una tabla grande que se puede guardar en el cassette para su posterior aprovechamiento.

De una manera muy breve vamos a indicar un posible programa para realizar este aprovechamiento.

Se trata de realizar un programa que saque el promedio de las columnas de una tabla de textos que contienen números o textos con espacios en blanco.

El tipo de promedio que hay que realizar es sobre los valores distintos de blanco que aparecen en las columnas de la tabla.

El programa se articula sobre un bucle que pide el nombre de un archivo que reside en cassette, lo carga y realiza los promedios.

Para finalizar se introduce la tecla STOP en la petición de nombre; de esta manera es válido cualquier nombre, incluso el nombre vacío, ya que sirve para cargar el siguiente fichero que se encuentra en el cassette tal como se hace en los programas.

El esquema de este programa es:

```
 10 REM dimensionado de la tabla
 20 LET tf = 30 : REM Número de filas.
 30 LET tc = 15 : REM Número de columnas.
 40 LET ac = 7  : REM Tamaño de los textos de cada casilla
 50 DIM t$(tf,tc,ac)
 60 LET b$=""
 70 IF LEN(b$) < ac THEN LET b$=" "+b$ : GO TO 70
 80 DIM c(tc) : DIM p(tc) : REM Listas auxiliares

100 REM Programa de cálculo de promedios
110 INPUT "Nombre de los datos:";n$
```

```
120 PRINT " Sitúe la cinta del cassette"
130 PRINT "Pulse la tecla PLAY del cassette"
140 LOAD n$ DATA t$()
150 PRINT " Pare el cassette"
160 GO SUB 1000 : REM Enviar a hacer los cálculos
170 GO TO 100
```

Desde la línea 10 hasta la 50 se definen las dimensiones de la tabla que debe estar de acuerdo con las definiciones dadas en el programa del editor de tabla.

Las líneas 60 y 70 preparan una variable *b$* que sólo contiene blancos y tiene la misma longitud que el elemento de la tabla. Así, para hacer la comparación de un elemento con un blanco no es necesario eliminar blancos ni añadirlos. Recuerde que el ZX-Spectrum tiene la longitud fija de los elementos de una tabla de textos.

La línea 110 pide el nombre del fichero de datos a leer. Como ya hemos dicho se puede entrar la cadena vacía para cargar la siguiente tabla del cassette.

Las líneas 120 y 130 nos indican qué operaciones debemos realizar para cargar esta tabla.

La línea 140 carga el fichero desde el cassette. El formato es el que requiere el ZX-Spectrum con la palabra DATA seguida de la tabla que se quiere rellenar.

La línea 150 nos indica que debemos parar el cassette una vez finalizada la carga.

La línea 160 envía el programa a hacer cálculos y finalmente se recicla a la línea 100 para leer otro nombre de un fichero de datos. Como ya hemos dicho, para finalizar hay que entrar un STOP (sin las comillas que coloca el ZX-Spectrum) en el momento de hacer la entrada del nombre.

Este esquema es el que se debe seguir para aprovechar los datos que hemos grabado en el cassette. A partir de aquí el programa es la excusa para hacer este planteamiento.

El mecanismo para sacar los promedios de las columnas es bastante sencillo, es necesario recorrer todas las columnas, dentro de cada columna recorrer toda fila, acumular y contar las casillas que no son blancos, y colocar el resultado en una tabla para cada una de las columnas.

Después de hacer estos cálculos se escriben en pantalla y el problema está resuelto.

El programa es el siguiente:

```
1000 REM Subrutina de hacer los promedios
1010 REM Recorrido de las columnas
1020 FOR i = 1 TO tc
1030     LET suma = 0 : LET cuenta=0
1040     FOR j = 1 TO tf
1050         IF t$(j,i) = b$ THEN GO TO 1080
1060             LET suma = suma + VAL ( t$(j,i) )
1070             LET cuenta = cuenta + 1
1080             NEXT j
1090     LET p(i)= suma : LET c(i)=cuenta
1100     NEXT i
```

```
2000 REM Fase de escritura
2010 CLS
2020 PRINT "Col.   Número   Promedio"
2030 PRINT "---------------------------"
2040 FOR i = 1 TO tc
2050      PRINT i;TAB(8);c(i);
2060      IF c(i) <> 0 THEN PRINT TAB(15);p(i)/c(i);
2070      PRINT
2080      NEXT i
2090 PRINT "---------------------------"
2100 RETURN
```

Se distinguen dos fases, la que se inicia en la línea 1010 que es el cálculo y la que se inicia en la línea 2000 que es la escritura de resultados.

En la fase de cálculo hay un primer bucle que recorre todas las columnas. Se inicia en la línea 1020 y acaba en la línea 1100.

En el interior del bucle se coloca la variable *suma* y la variable *cuenta* a cero, pues se inicia una nueva columna.

Se recorre mediante el bucle que empieza en la línea 1040 y finaliza en la línea 1080 en cada uno de los elementos que componen la columna.

El cuerpo del bucle consiste en una instrucción IF que filtra, es decir, no considera aquellos elementos de la columna que son blancos. Se utiliza la constante definida más arriba como *b$*.

Si el elemento no es blanco se acumula en suma su valor y se incrementa la cuenta en uno (líneas 1060 y 1070).

Una vez salidos de este recorrido se memoriza el resultado de la suma y de la cuenta en las tablas *p* y *c*. No se hace el promedio aún porque es posible que alguna columna no tenga ningún valor y se provoca una división por cero.

En la fase de presentación de resultados se inicia limpiando la pantalla. A continuación se imprime una cabecera que consta de tres partes: de qué columna se trata, de cuántos elementos se compone distintos de blanco y el valor del promedio si existe; es decir, si hay por lo menos un elemento distinto de blanco.

De la línea 2040 hasta la 2080 se utiliza un bucle para considerar cada una de las columnas.

En la línea 2050 se imprime de qué columna se trata al inicio y en columna 9 el valor de la cuenta para la columna.

En la línea 2060 se pregunta si la cuenta es distinta de cero, en caso afirmativo se imprime en la columna 16 el cociente del resultado acumulado partido por el número de elementos.

Observe que todas las instrucciones PRINT anteriores finalizan con un punto y coma. La línea 2070 sirve para saltar a la línea siguiente de pantalla.

Finalmente la línea 2090 imprime un pie de tabla que consiste en una línea de guiones.

Para terminar hemos utilizado dos tablas que hay que definir, la *p* y la *c*.

La línea 80 realiza esta función:

```
80 DIM c(tc) : DIM p(tc)
```

hay que colocar esta instrucción en la cabecera porque sólo es necesario ejecutarla antes de empezar el programa y, por lo tanto, no debe colocarse en el interior de la subrutina.

Para probar el programa utilice la práctica de la lección anterior para rellenar una tabla que sólo contiene elementos en las tres primeras columnas. La primera columna tiene todos los valores llenos; es decir, 1, 2, 3, 4 hasta 30. La segunda sólo tiene los 20 primeros valores llenos; es decir, 1, 2, 3, 4 hasta 20. Y la tercera columna sólo tiene los 10 primeros valores llenos.

En cada elemento se coloca el número correspondiente a la fila al que pertenece.

Los resultados que debe obtener son:

Col.	**Número**	**Promedio**
1	30	15.5
2	20	10.5
3	10	5.5
4	0	
5	0	
....		
15	0	

Capítulo 14

ESQUEMA DE CONTENIDO

Caracteres gráficos	
Gráficos de alta resolución	Operaciones primitivas Operaciones auxiliares Pendiente de las líneas rectas
Gráficos bidimensionales	Trazados de polígonosy circunferencias Elipses y espirales Figuras de Lissajous
Diagramas	Diagramas de barras Diagrama circular Diagramas X-Y
Superposición de líneas	Borrado de líneas Dibujo interactivo

14.1 CARACTERES GRAFICOS

En el ZX-SPECTRUM los caracteres ocupan un cuadrado que está dividido en cuatro zonas. Por tanto existen 16 caracteres gráficos. Sus códigos van desde el 128 (cuadrado en blanco) hasta el 143 (cuadrado completamente en negro) como se ve en la figura 1. Para obtener estos símbolos emplearemos la función CHR$. Por ejemplo:

```
PRINT CHR$(137)
```

Los caracteres gráficos se pueden obtener con *CHR$* o desde el teclado

En el ZX-SPECTRUM también se pueden obtener estos símbolos directamente del teclado. En primer lugar hay que poner el teclado en modo gráfico. Para ello se mantiene apretada CAPS SHIFT y luego se teclea GRAPHICS (sobre el 9) (si su ordenador es el ZX-SPECTRUM normal) o se pulsa simplemente la tecla GRAPH (si su ordenador es el ZX-SPECTRUM+). Observamos que en el cursor aparece la letra G. Para salir del modo gráfico repetiremos exactamente la misma operación. Los caracteres gráficos están sobre las teclas numéricas (del 1 al 8). Si estando el cursor en modo G pulsamos una de estas teclas, por ejemplo el 3, aparece el símbolo gráfico correspondiente que es una raya horizontal. Si pulsamos de nuevo la tecla 3 manteniendo pulsada CAPS SHIFT, entonces aparece el símbolo inverso, es decir, en blanco lo que antes era negro y viceversa.

El programa siguiente imprime una tabla de cuadrados utilizando los caracteres gráficos para separar las columnas.

Programa de «Tabla de cuadrados»

```
10 FOR I=1 TO 14:PRINT CHR$(143);:NEXT I
20 PRINT
30 LET A$=CHR$(138): LET B$=CHR$(133)
40 FOR I=1 TO 15
50   PRINT A$;" ";I;TAB(6);A$;" ";
           I*I;TAB(13);B$
60   NEXT I
```

La línea 10 escribe una línea horizontal. La línea 20 hace empezar una nueva línea. En la línea 30 almacenamos los caracteres gráficos 138 y 133 a fin de simplificar la escritura de la línea 50. Las funciones CHR$(138) y CHR$(133) se pueden cambiar por los símbolos gráficos obtenidos directamente del teclado escritos entre comillas. La línea 40 empieza el bucle de la tabla. En la siguiente línea se escribe un signo vertical (contenido de A$), el valor de la variable I, otro símbolo vertical, el cuadrado de I y otro símbolo vertical contenido en B$. Finalmente, la línea 60 cierra el lazo de la línea 40. Si quiere cerrar el cuadrado puede añadirle las mismas instrucciones de la línea 10 con el 70 como número de línea.

El siguiente programa es un ejemplo de cómo pueden construirse sím-

bolos de tamaño superior al normal. Se han escogido solamente 4 dígitos (de 0 a 3) como símbolos a representar a fin de no alargar el listado.

Programa de «Dibujo de números»

```
 10 DIM A$(4,6,1)
 20 FOR I=1 TO 4
 30   FOR J=1 TO 6
 40       READ G: LET A$(I,J)=CHR$(128+G)
 50       NEXT J
 60   NEXT I
 70 INPUT "CIFRA (0-3):";C
 80 IF C<0 THEN STOP
 85 LET I=C+1
 90 PRINT A$(I,1);A$(I,2)
100 PRINT A$(I,3);A$(I,4)
110 PRINT A$(I,5);A$(I,6)
115 PRINT
120 GOTO 70
130 DATA 11,7,10,5,14,13
140 DATA 9,10,0,10,4,14
150 DATA 3,7,4,2,14,12
160 DATA 3,7,4,13,12,13
```

En la línea 10 se define una tabla de cuatro elementos, cada uno de los cuales tiene 6 textos de un solo símbolo. En el bucle comprendido entre 20 y 60 se llena esta tabla, a partir de los datos contenidos en las instrucciones 130 a 160. Para simplificar, los caracteres gráficos se han almacenado restándoles 128 a cada uno. En la segunda instrucción de la línea 40, se suma de nuevo 128, con lo cual nos da de nuevo el número del carácter respectivo. En la línea 70, el programa nos pide qué cifra queremos que se imprima. Las líneas 90, 100 y 110 imprimen la cifra escogida y el control pasa de nuevo a la línea 70. Para detener el programa se entra un número negativo. Entonces el IF de la línea 80 detiene el proceso.

Combinando caracteres gráficos podemos obtener símbolos ampliados

Este programa representará en pantalla la cifra que le pidamos en tamaño ampliado. En este momento sólo se admiten cuatro posibilidades (de 0 a 3) pero se puede ampliar fácilmente para que escriba cualquier otro carácter.

Observe que en las líneas 130 a 160 se podrían haber entrado directamente los códigos correspondientes a cada símbolo. Así, por ejemplo, la línea 130, que almacena los datos para dibujar el cero, podríamos haberla escrito así:

```
130 DATA 139,135,138,133,142,141
```

En este caso, no se olvide de borrar de la línea 40 el 128+, dejando únicamente en el paréntesis la variable G.

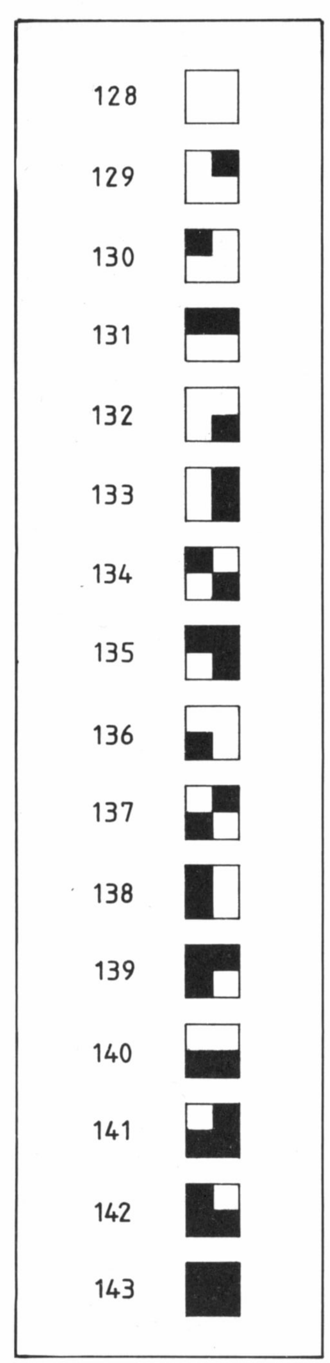


Figura 1. Conjunto de caracteres gráficos utilizables en el ZX-SPECTRUM

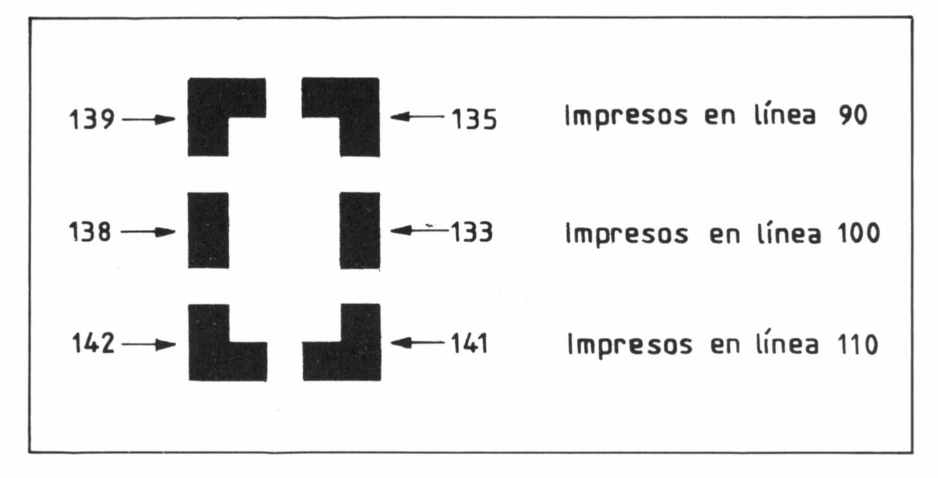


Figura 2. Conjunto de caracteres gráficos que se utilizan para dibujar el cero.

Después de estos cambios ejecute el programa y en la cifra pedida déle para que escriba el cero.

Para que vea cómo está construido el cero, le presentamos la figura 2 y el número de cada símbolo. En la figura se han separado ligeramente para que pueda observar mejor los símbolos que lo integran. Vea también el orden en que están los datos en la nueva línea 130 y su correspondencia en la figura.

Como ejercicio, modifique usted las líneas 140 a 160 al igual que hemos hecho con la 130. Ya sabe que lo único que tiene que hacer es sumar 128 a los números que aparecen en estas líneas.

Ahora tome papel cuadriculado e intente dibujar los demás números, añadiendo una nueva línea de DATA y modificando el 4 que aparece en la instrucción DIM de la línea 10 por un 5 y en la línea 20, si añade otra línea y así por cada nueva línea que añade. En la figura 3 le presentamos el diseño del número 4. Recuerde que siempre tiene que utilizar seis símbolos.

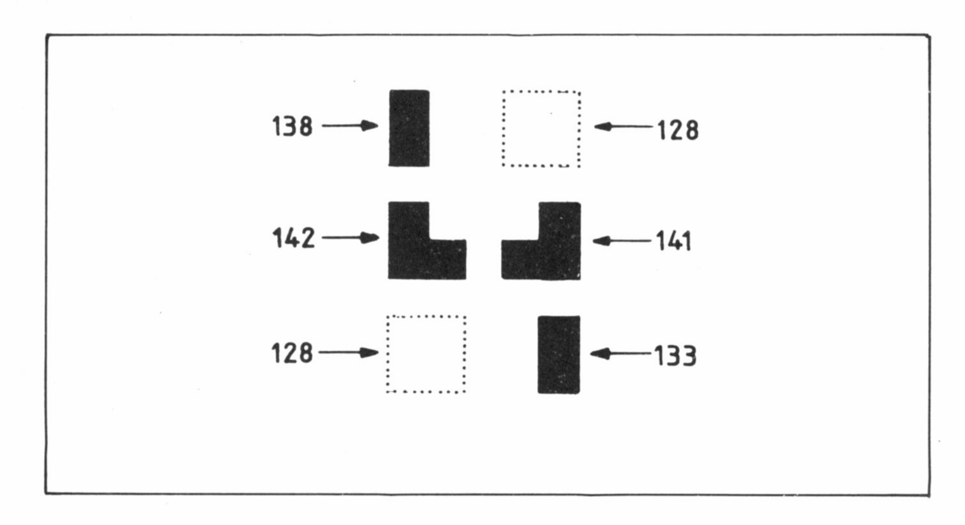


Figura 3. Conjunto de caracteres gráficos que se utilizan para dibujar el cuatro.

Programa de «Dibujo cuadrado»

Al realizar el programa del texto general que estudiará en seguida en el capítulo estándar debe cambiar las líneas 50, 60 y 80 por las siguientes, para que el programa funcione en el ZX-SPECTRUM.

Téngalo presente.

```
50 FOR I=1 TO 11
60 PRINT AT I,0;CHR$(138);AT I,11;CHR$(133);
80 PRINT AT 12,0;CHR$(142);
```

14.2

14.2 GRAFICOS DE ALTA RESOLUCION

La pantalla, o para ser más exactos, la zona de presentación de la pantalla en el ZX-SPECTRUM tiene una resolución de 176×256. El eje *Y* va de cero a 175 y el eje *X* de cero a 255. Las dimensiones son casi el doble de las que empleamos en el texto general por lo que la mayoría de programas requerirán una ampliación de las dimensiones a fin de aprovechar la pantalla en su totalidad.

14.2.1 Operaciones primitivas

En el ZX-SPECTRUM existen algunas particularidades respecto a las operaciones primitivas.

a) Posicionamiento de un pixel:

En el ZX-SPECTRUM, la instrucción para activar un pixel tiene la forma

```
PLOT X,Y
```

La palabra PLOT («plot» significa grafiar en inglés) se encuentra sobre la tecla Q. Esta instrucción cumple con las especificaciones asignadas a la operación de colocar un punto. Los valores que se dan a X y a Y son los que definen la coordenada donde se coloca el punto. Así PLOT 255, 175 colocaría un punto en el extremo superior derecho de la pantalla.

b) Trazado de una línea:

Para trazar una línea, se utiliza la instrucción

```
DRAW N,M
```

Esta instrucción es algo especial. Los valores de N y M no son las coordenadas del punto final, sino que son los incrementos que hay que añadir a las coordenadas del punto actual. Por tanto, los valores de N y M pueden ser negativos.

La instrucción DRAW trazalíneas con desplazamiento relativo

La palabra DRAW («draw» significa dibujar en inglés) se encuentra sobre la tecla W. Ejemplos:

```
PLOT 10,10: DRAW 50,50
PLOT 100,50:DRAW 50,50
```

En el primer caso se traza una recta desde (10, 10) hasta (60, 60). En el segundo caso la recta va desde (100, 50) a (150, 100).

c) Posicionamiento de un carácter

No existe esta operación en el ZX-SPECTRUM. En su lugar emplearemos la función AT. Para relacionar la posición del carácter con la del pixel, hay que tener en cuenta que cada carácter ocupa un recuadro de 8×8 pixels.

La instrucción DRAW se aparta de las especificaciones que veremos para la instrucción LINEA en la lección estándar. En algunas ocasiones, la instrucción DRAW será suficiente para nuestros fines. Cuando ello no sea posible, utilizaremos un par de subrutinas que simularán el funcionamiento de las instrucciones PUNTO y LINEA. Estas subrutinas son:

```
 999 REM Punto
1000 PLOT X,Y
1010 LET XA=X: LET YA=Y
1020 RETURN
```

```
1999 REM Linea
2000 LET DX=X-XA: LET DY=Y-YA
2010 DRAW DX,DY
2020 LET XA=X: LET YA=Y
2030 RETURN
```

Es decir, cuando usted encuentre en un programa la instrucción

```
PUNTO(X,Y)
```

para que se realice, en el ZX-SPECTRUM usted debe escribir en su lugar un acceso (GOSUB) a la subrutina PUNTO, que abarca las líneas vistas de la 1000 a la 1020. Si lo que encuentra en el programa principal es:

```
LINEA(X,Y)
```

deberá utilizar en su lugar un acceso a la subrutina LINEA que abarca las líneas vistas de la 2000 a la 2030.

Escribiendo estas subrutinas al final del programa, bastará que en el programa principal remita a ellas mediante un GOSUB, seguido del número de línea de la subrutina correspondiente.

Lógicamente la numeración puede ser alterada si es necesario. En la subrutina 1000 se activa el pixel de coordenadas X e Y. Además, las coordenadas del punto actual se almacenan en XA e YA. En la subrutina 2000, se pretende trazar una recta desde el punto actual hasta X, Y. Para ello se calculan los incrementos DX y DY. Se traza la recta y se almacenan las nuevas coordenadas del punto actual.

El siguiente programa constituye un ejemplo de utilización de estas subrutinas.

```
  10 LET X=100: LET Y=50: GOSUB 1000
  20 LET X=150: LET Y=100: GOSUB 2000
  30 STOP
 999 REM Punto
1000 PLOT X,Y
1010 LET XA=X: LET YA=Y
1020 RETURN
1999 REM Linea
2000 LET DX=X-XA: LET DY=Y-YA
2010 DRAW DX,DY
2020 LET XA=X: LET YA=Y
2030 RETURN
```

Las instrucciones 10 y 20 trazan una recta de (100, 50) hasta (150, 100).

Las operaciones primitivas las realizaremos con dos subrutinas

Este par de subrutinas se supondrán siempre presentes en los programas que realicemos a partir de ahora. No se incluirán en los listados de los programas de esta lección a fin de no alargarlos demasiado, aunque insistimos que deberán estar presentes en los programas reales. Puede ser interesante tenerlas grabadas en una cinta y cargarlas cada vez que sea necesario con LOAD o MERGE. También las puede copiar en un papel aparte para tenerlas a mano sin tener que volver a buscarlas en el texto.

Para utilizar estas subrutinas hay que tener presente que las variables de comunicación con el programa principal son X e Y. Además, debemos tener la precaución de no utilizar en el programa principal, las variables que son internas de las subrutinas.

Estas variables son XA, YA, DX y DY.



14.2.2 Operaciones auxiliares

El ZX-SPECTRUM incorpora una operación auxiliar para trazar círculos. La instrucción tiene la siguiente forma general:

```
CIRCLE X,Y,R
```

La palabra CIRCLE («circle» significa círculo en inglés) se obtiene pulsando en modo E la tecla SYMBOL SHIFT y la tecla H simultáneamente. Los valores de X e Y son las coordenadas del centro y R es el valor del radio. Por ejemplo:

```
CIRCLE 130,80,70
```

Esta instrucción dibuja un círculo en las coordenadas (130, 80) que aproximadamente coinciden con el centro de la pantalla, con un radio de 70.

La instrucción DRAW sirve también para trazar líneas curvas

Para trazar arcos de circunferencia se puede utilizar una variante de la instrucción DRAW. Si se especifica un tercer número en la instrucción DRAW, entonces en lugar de una línea recta, traza un arco. El tercer número especifica el número de radianes que debe girarse. Por tanto, este tercer número ha de ser siempre inferior o igual a 3,1416, que se obtiene pulsando PI, situada en la tecla de la letra M.

Ejemplo:

```
PLOT 100,100: DRAW 50,50,PI
```

Se dibuja un semicírculo entre (100, 100) y (150, 150).

14.2.3 Pendiente de las líneas rectas

En el programa que se utiliza para dibujar rectas de varias pendientes no hace falta emplear las dos subrutinas anteriores. En este caso es suficiente cambiar las líneas 30 y 60 por

```
30 PLOT 0,0
60 DRAW X,Y
```



Tenga esto presente cuando realice el programa «Pendientes de líneas rectas» en el capítulo estándar. Esto es debido a que todas las rectas parten de (0, 0) y precisamente éste es el único caso en el que la instrucción DRAW es equivalente a LINEA.

14.3 GRAFICOS BIDIMENSIONALES

14.3.1 Trazado de polígonos y circunferencias

Para trazar círculos en el ZX-SPECTRUM, en la mayoría de casos lo más práctico es utilizar la instrucción CIRCLE. Podría ser interesante superponer dos círculos cada uno hecho por un método distinto y ver las posibles diferencias. Si queremos trazar un círculo con la máxima precisión que permite la resolución de la pantalla, debemos construirlo punto a punto. El programa quedaría entonces:

```
10 LET XO=130: LET YO=87: LET R=80
20 FOR A=0 TO 6.284 STEP 0.01745
30   LET X=XO+R*COS(A)
40   LET Y=YO+R*SIN(A)
50   PLOT X,Y
60   NEXT A
```

La instrucción *CIRCLE* es más rápida pero menos precisa que si se utiliza *SIN* y *COS*

El 6.284 que aparece en la línea 20 no es otra cosa que el valor numérico de la longitud de la circunferencia, es decir 2π ($2\times3{,}1416$).

Si ejecutamos este programa veremos que el círculo trazado es más preciso que el obtenido con la instrucción CIRCLE pero en cambio es mucho más lento. La superposición de los dos círculos es muy fácil de hacer añadiendo la siguiente línea al programa.

```
70 CIRCLE XO,YO,R
```

Uno de los objetivos de los programadores es aumentar la velocidad de los programas. Una manera de conseguirlo es eliminar las operaciones superfluas. Podría parecernos que en el programa anterior no hay ninguna operación que esté de más ya que es obligado calcular todos los puntos de la circunferencia. Si bien esto es cierto, en realidad no hace falta emplear forzosamente las funciones SIN y COS para calcular absolutamente todas las coordenadas. Estas dos funciones son costosas de cálculo como hemos podido comprobar.

Existe un método para trazar círculos calculando un solo cuadrante. Los puntos de los otros tres cuadrantes se obtienen por simples desplazamientos. (Repasemos los conocimientos de trigonometría). El programa queda:

```
10 REM Circulo
20 LET XO=130: LET YO=80: LET R=70
30 FOR A=0 TO 1.571 STEP 0.01745
40   LET X=R*COS(A)
```

```
50    LET Y=R*SIN(A)
60    PLOT XO+X,YO+Y: PLOT XO-X,YO+Y
70    PLOT XO+X,YO-Y: PLOT XO-X,YO-Y
80    NEXT A
```

El bucle de la línea 30 va desde cero hasta π/2 radianes (un cuadrante), o lo que es lo mismo, 6.284 que es la longitud de la circunferencia dividido por 4, puesto que se trata de calcular un cuadrante. En las líneas 60 y 70 se dibujan los puntos de los cuatro cuadrantes. Puesto que el ordenador calcula mucho más rápidamente una suma o una resta que una función SIN o COS, este programa es bastante más rápido que el anterior como se puede observar fácilmente.

Programa de «Trazado de polígonos»

En el programa que emplearemos en la lección estándar para trazar polígonos es posible ampliar, si se desea, el tamaño de la figura a fin de aprovechar las mayores dimensiones de la pantalla del ZX-SPECTRUM. Entonces la línea 50 quedará:

```
50 LET XO=130: LET YO=80: LET R=70
```

Por otra parte, hay que sustituir las instrucciones de las líneas 60 y 100 por sus equivalentes en el ZX-SPECTRUM. Las instrucciones quedan:

```
 60 LET X=XO+R: LET Y=YO: GOSUB 1000
100 GOSUB 2000
```

En donde las subrutinas 1000 y 2000 son las que se indicaron anteriormente y que hay que añadir al listado. Tenga abierto el capítulo de prácticas al escribir el programa de la lección estándar (color azul) para ir añadiendo estos cambios.

Por un error de diseño, la instrucción DRAW cuando recibe consecutivamente coordenadas prácticamente iguales, es incapaz de dibujar correctamente y se comporta de forma errática. Este hecho lo podemos ver si utilizamos el programa anterior para construir un polígono de muchísimos lados (más de 200).

Programa de «Trazado de círculos»

En el programa para trazar círculos hay que cambiar las líneas 30 y 70 por

14.4

```
30 LET X=XO+R: LET Y=YO: GOSUB 1000
70 GOSUB 2000
```

14.3.2 Elipses y espirales

En el programa empleado para *dibujar elipses* en el capítulo estándar (color azul), efectuaremos la modificación de las líneas 40 y 80. Estas líneas quedan:

```
40 LET X=XO+R1: LET Y=YO: GOSUB 1000
80 GOSUB 2000
```

Además, le añadiremos las subrutinas 1000 y 2000 explicadas anteriormente.

En el *programa que dibuja espirales,* cambiaremos los valores de las coordenadas XO e YO (línea 20) para que la espiral quede centrada en la pantalla. Por otra parte, cambiaremos las instrucciones PUNTO y LINEA por sus equivalentes en el ZX-SPECTRUM. Las modificadas quedan:

```
20 LET XO=130: LET YO=80: LET R=20
50 LET X=XO+R: LET Y=YO: GOSUB 1000
90 GOSUB 2000
```

14.3.3 Figuras de Lissajous

Al realizar este programa en el capítulo del texto estándar tenga en cuenta estas observaciones. Al programa indicado en el texto hay que añadirle las dos subrutinas 1000 y 2000. Entonces las líneas 60 y 100 quedan:

```
 60 LET X=XO: LET Y=YO+YM*COS(-R): GOSUB 1000
100 GOSUB 2000
```

Por otra parte podemos ampliar el gráfico a fin de aprovechar el tamaño total de la pantalla del ZX-SPECTRUM. Para ello modificaremos las líneas 20 y 30 que quedarán:

```
20 LET XO=130: LET YO=80
30 LET XM=70: LET YM=70
```



14.4 DIAGRAMAS

14.4.1 Diagramas de barras

Tenga en cuenta estas modificaciones en los programas que realizará: En el programa indicado en el texto general para dibujar el diagrama de

barras se pueden sustituir las instrucciones gráficas por llamadas a las dos subrutinas 1000 y 2000 que ya conocemos. Sin embargo, puesto que este tipo de diagrama es muy sencillo ya que todo son líneas rectas, emplearemos directamente la instrucción DRAW. Entonces, las subrutinas 300 y 400 quedan, después de ampliar el tamaño para adaptarlo a la pantalla del ZX-SPECTRUM:

```
300 REM ejes
310 PLOT 20,150
320 DRAW 0,-130
330 DRAW 200,0
340 RETURN
```

```
400 REM Barras
410 PLOT P+20,20
420 DRAW 0,L
430 DRAW 20,0
440 DRAW 0,-L
450 RETURN
```

Para dibujar las barras con *rayado interior,* introduciremos las modificaciones indicadas en el texto de la lección estándar con estas adaptaciones al SPECTRUM. La línea 20 queda:

```
20 LET N=5: DIM A(N): DIM R(N)
```

Como antes, emplearemos la instrucción DRAW directamente en las líneas 444 y 446 que quedan:

```
444 PLOT P+20,J
446 DRAW 20,0
```

14.4.2 Diagrama circular

Para realizar el diagrama circular hay que añadir las subrutinas 1000 y 2000 al programa indicado en el texto general. En la subrutina 300 de este programa introduciremos los siguientes cambios en las líneas 310 y 350:

```
310 LET X=XO: LET Y=YO: GOSUB 1000
350 GOSUB 2000
```

En el texto estándar se indica un programa que permite realizar la extracción de un sector. En ese programa cambiaremos las líneas 365 y 450 por:

```
365 LET X=X1: LET Y=Y1: GOSUB 2000
450 LET X=X1: LET Y=Y1: GOSUB 1000
```

aparte de las modificaciones ya indicadas para las líneas 310 y 350.

En el ZX-SPECTRUM se dispone de la operación auxiliar CIRCLE, cuyo uso puede simplificar este programa. Las líneas comprendidas entre la 10 y la 70 del programa original para dibujar círculos no sufren variación. A partir de ahí el programa queda, borrando la 75.

```
 80 CIRCLE XO,YO,R
 90 FOR I=1 TO N
100   LET RA=RA+6.284*A(I)/S
110   LET X=R*COS(RA)
120   LET Y=R*SIN(RA)
130   PLOT XO,YO: DRAW X,Y
140   NEXT I
150 STOP
160 DATA 9,12,6,18,6
```

La línea 80 traza el círculo y el bucle iniciado en la línea 90 dibuja los radios que delimitan los sectores circulares. El programa termina en la 160 y no existen subrutinas.

CIRCLE no sirve para extraer sectores

Este programa tiene la ventaja de la simplificación pero, en cambio, no permite realizar la extracción de sectores.



14.4.3 Diagramas X–Y

En el programa utilizado pàra *grafiar funciones* realizaremos los siguientes cambios. Las líneas 20, 30 y 40 quedan:

```
30 PLOT 120,160: DRAW 0,-160
40 PLOT 0,80: DRAW 240,0
```

Se ha ampliado el tamaño de los ejes a fin de aprovechar mejor la pantalla. Por la misma razón, en las líneas 100 y 110 hay que cambiar el 50 por un 80 y en la línea 70, el final del bucle puede ser un 240 en lugar

de un 200. Conviene resaltar que estas modificaciones sólo afectan al tamaño del gráfico y no es obligatorio realizarlas. Por el contrario, sí que es obligatorio modificar las líneas 120 y 130.

```
120 IF A=0 THEN GOSUB 1000: LET A=1: GOTO 140
130 GOSUB 2000
```

Podemos probar alguna otra función, aparte de las indicadas en el texto, como:

```
50 DEF FNA(X)=10*EXP(X/9)*SIN(X) con S=16
```

```
50 DEF FNA(X)=X/(0.1*X+1) con S=9
```

El ZX-SPECTRUM en la instrucción *DIM* sólo admite una variable

En el programa utilizado para *grafiar puntos* introduciremos, las siguientes modificaciones:

```
 20 LET N=5: DIM A(N): DIM B(N)
160 GOSUB 1000
190 GOSUB 2000
```

Se le puede añadir también un par de instrucciones para trazar los ejes de coordenadas. Estas instrucciones serán:

```
25 PLOT 0,160: DRAW 0,-160: DRAW 250,0
```

14.5.3

14.5 SUPERPOSICION DE LINEAS

Para indicar que se escribe en modo inverso se utiliza la palabra INVERSE que se encuentra escrita en la tecla M. La palabra INVERSE se puede usar para gráficos y también para caracteres normales. Detrás de la palabra INVERSE se indica un número que puede ser un 1 o un 0 según se escriba en blanco o en negro.

Por ejemplo:

```
PRINT INVERSE 1,"HOLA"
```

La opción INVERSE también afecta a la instrucción PRINT

La palabra HOLA aparece escrita en blanco sobre fondo negro. Este tipo de escritura nos será útil para imprimir cabeceras o cualquier texto que deba destacarse. Aquí la utilizaremos solamente para los gráficos. Las dos instrucciones básicas, cuando se utilizan en modo inverso se escriben:

```
PLOT INVERSE 1,X,Y
DRAW INVERSE 1,X,Y
```

Para indicar superposición se emplea la palabra OVER que se encuentra escrita en la tecla N. OVER significa encima en inglés. Al igual que la palabra INVERSE, la palabra OVER va seguida de un número. Las dos instrucciones básicas cuando se utilizan en modo superposición se escriben:

```
PLOT OVER 1,X,Y
DRAW OVER 1,X,Y
```

La escritura con superposición también se admite en una instrucción PRINT. Sirve para superponer dos caracteres formando uno solo (caso típico de los acentos). Por ejemplo:

```
10 OVER 1
20 PRINT "o";CHR$(8);"'"
```

Con la función OVER se pueden superponer caracteres

Este programa escribirá una *o* minúscula acentuada. La línea 10 coloca la pantalla en modo superposición. La instrucción PRINT escribe una *o,* a continuación el valor 8 para hacer retroceder una posición al cursor y entonces se escribe el acento de modo que quede superpuesto sobre la letra *o*.

14.5.1 Borrado de líneas

El programa de borrado de líneas se escribirá de la siguiente forma en su ordenador:

```
10 PLOT 0,0
20 DRAW 250,175
30 INPUT A$
40 PLOT INVERSE 1,0,0
50 DRAW INVERSE 1,250,175
```

Al escribir RUN se traza una recta en diagonal y el programa se detiene en el INPUT de la línea 30. Pulsando ENTER, se borra la línea. Si invertimos el sentido de borrado, es decir, cambiando

```
40 PLOT INVERSE 1,250,175
50 DRAW INVERSE 1,-250,-175
```

14.6

entonces el borrado es imperfecto quedando puntós sueltos en la pantalla.

14.5.2 Dibujo interactivo

Realizaremos algunas pequeñas modificaciones del programa del texto general para adaptarlo al ZX-SPECTRUM. En el programa principal las modificaciones son mínimas pues los códigos de las flechas coinciden con los indicados. Unicamente, si se desea, se puede ampliar el tamaño de la pantalla utilizable escribiendo:

```
20 LET XM=200: LET YM=150: LET P=0
```

También es conveniente utilizar una instrucción PAUSE antes de la función INKEY$ como ya vimos en su momento. La línea 40 queda:

```
40 PAUSE 0: LET T$=INKEY$
```

Donde sí aparecen las modificaciones importantes es en las subrutinas. Las líneas 210, 310, 410 y 510 tendrán todas la forma:

```
IF P=0 THEN PLOT INVERSE 1,X,Y
```

Asimismo, las líneas 230, 330, 430 y 530 tendrán todas la forma:

```
PLOT X,Y
```

Al final del programa se añadirán las conocidas subrutinas 1000 y 2000. Entonces, las subrutinas 600 y 700 quedan:

```
600 LET P=1
610 GOSUB 1000
620 RETURN
```

```
700 LET P=1
710 PLOT XA,YA
```

```
720 GOSUB 2000
730 RETURN
```

En este momento, el programa ya es operativo.

Al escribir RUN (y fin de línea) aparecerá un punto en el centro de la pantalla. Con las flechas de movimiento del cursor desplazaremos el punto en la dirección que deseemos.

La opción *INVERSE* la utilizamos para borrar líneas

Si queremos añadir la operación de borrado de líneas, el listado de la subrutina 800 es el siguiente:

```
800 PLOT INVERSE 1,XA,YA
810 DRAW INVERSE 1,X-XA,Y-YA
820 RETURN
```

No olvide añadir también una instrucción IF en la línea 110 del programa principal.

Otra operación que sería interesante añadir al programa es el trazado de circunferencias. Esta operación funcionará del siguiente modo. Al pulsar una C, el programa pedirá el valor del radio. Una vez el operador haya contestado, trazará la circunferencia centrada en las coordenadas del punto y con el radio especificado. En el programa principal añadiremos la línea

```
120 IF T$="C" OR T$="c"THEN GOSUB 900:GOTO 40
```

El listado de la subrutina 900 será el siguiente:

```
900 INPUT "RADIO:",R
910 CIRCLE X,Y,R
920 PLOT X,Y
930 RETURN
```

La instrucción PLOT de la línea 920 es necesaria puesto que CIRCLE modifica las coordenadas del punto actual.

14.6.2

Capítulo 15

ESQUEMA DE CONTENIDO

Operaciones bidimensionales	Memorización de coordenadas Traslación Escalado Rotación
Dibujos tridimensionales	Representación de funciones
Gráficos en color	Aplicaciones
Memorización de pantalla	Instrucción POINT Memorización global
Sonido	

15.1 OPERACIONES BIDIMENSIONALES

En el ZX-SPECTRUM existen otras posibilidades gráficas aparte de las comentadas en este capítulo, pero para utilizarlas hay que emplear las instrucciones PEEK y POKE que se verán en un capítulo posterior. Algunos casos particulares, como por ejemplo la animación (dibujos con movimiento) se reservan para un estudio que se efectuará más adelante.

En este capítulo utilizaremos de nuevo las subrutinas 1000 y 2000 para sustituir a las instrucciones convencionales PUNTO y LINEA, cuyos listados recordamos a continuación:

```
 999 REM Punto
1000 PLOT X,Y
1010 LET XA=X: LET YA=Y
1020 RETURN
```

```
1999 REM Linea
2000 LET DX=X-XA: LET DY=Y-YA
2010 DRAW DX,DY
2020 LET XA=X: LET YA=Y
2030 RETURN
```

15.1.1 Memorización de coordenadas

En el programa utilizado para dibujar cualquier tipo de figura en el capítulo estándar, cambiaremos la línea 20 por:

```
20 LET N=4: DIM A(N): DIM B(N)
```

Esto es debido, como sabemos, a que la instrucción DIM del ZX-SPECTRUM difiere del comportamiento estándar y sólo permite definir una variable cada vez. A continuación vienen las modificaciones correspondientes a las instrucciones gráficas.

Las líneas 510 y 540 quedan:

```
510 LET X=A(1): LET Y=B(1): GOSUB 1000
540 GOSUB 2000
```

Por supuesto, al final del programa se añadirán las subrutinas 1000 y 2000 que las obtendremos de una cinta donde previamente se hayan grabado, o bien se teclearán de nuevo. Por otra parte, recordemos que nuestra

pantalla tiene unas dimensiones de 255×175, que son algo mayores que las que se utilizan en el texto general. Esto nos permite dibujar figuras más grandes.



15.1.2 Traslación

Para el programa de traslación necesitaremos construirnos las operaciones equivalentes al borrado de un punto y de una línea.

Las instrucciones son PLOT INVERSE y DRAW INVERSE. Sin embargo, como ya sabemos, la instrucción DRAW no traza líneas en modo absoluto, sino que realiza desplazamientos relativos, es decir, que las coordenadas finales dependen del punto de partida. Puesto que este tipo de instrucción es incómodo para utilizar dentro de un programa, nos construiremos dos subrutinas que simulen el comportamiento de las instrucciones PUNTO y LINEA INVERSAS. Sus listados son:

```
1099 REM Punto inverso
1100 PLOT INVERSE 1,X,Y
1110 LET XA=X: LET YA=Y
1120 RETURN
```

```
2099 REM Linea inversa
2100 LET DX=X-XA: LET DY=Y-YA
2110 DRAW INVERSE 1,DX,DY
2120 LET XA=X: LET YA=Y
2130 RETURN
```

Estas dos subrutinas están construidas según el modelo empleado para las subrutinas 1000 y 2000, exceptuando la opción INVERSE 1 para las instrucciones PLOT y DRAW. La subrutina 1100 sirve para borrar un punto y la 2100 para borrar una línea. Si ya teníamos grabadas las subrutinas 1000 y 2000 sería conveniente añadirles estas dos y almacenar las cuatro juntas. En programas posteriores necesitaremos utilizar las cuatro subrutinas.

En el programa de traslación del texto estándar efectuaremos los siguientes cambios. La línea 20 queda:

```
20 LET N=13: DIM A(N): DIM B(N)
```

a causa de las razones apuntadas anteriormente. En la subrutina 500 modificaremos las líneas 510 y 540:

```
510 LET X=A(1): LET Y=B(1): GOSUB 1000
540 GOSUB 2000
```

Finalmente, en la subrutina 600, cambiaremos de forma análoga las líneas 610 y 640 por:

```
610 LET X=A(1): LET Y=B(1): GOSUB 1100
640 GOSUB 2100
```

Puesto que la pantalla es más amplia, dispondremos de un mayor margen de maniobra para mover la figura. No obstante, el ZX-SPECTRUM no dispone de la opción de recortado automático de figuras. En consecuencia, si intentamos traspasar los límites de la pantalla se producirá un error, con el mensaje:

B Integer out of range

que nos indica que usamos un número (un número entero) fuera de los márgenes correctos.

15.1.2

15.1.3 Escalado

Para este programa que veremos en el capítulo estándar aprovecharemos todas las subrutinas del caso anterior, incluyendo las de borrado, 1100 y 2100. De esta forma, las modificaciones son mínimas. Solamente hay que variar la instrucción DIM de la línea 20, quedando:

```
20 LET N=13: DIM A(N): DIM B(N)
```

El resto del programa principal funciona directamente sin introducir cambios.

En el programa para realizar el *efecto zoom,* modificaremos las siguientes líneas:

```
 20 DIM A(4): DIM B(4)
210 GOSUB 1000
240 GOSUB 2000
310 GOSUB 1100
340 GOSUB 2100
```

En este caso no se aprovechan las subrutinas 500 y 600, pero sí el grupo de cuatro que realizan las operaciones básicas. Por otra parte, puesto que la velocidad de cálculo es un factor crítico, se aconseja no dejar los espacios en blanco habituales para señalar el inicio y final de un bucle. El valor más apropiado para el límite del bucle de la línea 80 está comprendido entre 10 y 15. Sin embargo, en el ZX-SPECTRUM, la velocidad de cálculo no es lo bastante elevada como para apreciar completamente el efecto zoom.

15.1.4 Rotación

Para elaborar este programa aprovecharemos el conjunto de subrutinas del programa anterior. Además, hay que cambiar, como siempre, la línea 20 por:

```
20 LET N=4: DIM A(N): DIM B(N)
```

Para efectuar la conversión de grados a radianes, recordemos que el ZX-SPECTRUM dispone de la función PI (en la letra M) cuyo valor es precisamente el número π.

15.1.3

15.2 DIBUJOS TRIDIMENSIONALES

En el programa utilizado para representar *superficies tridimensionales* emplearemos las instrucciones PLOT y DRAW directamente. Utilizaremos estas instrucciones propias del ZX-SPECTRUM en lugar de recurrir a las conocidas rutinas 1000 y 2000, a causa de que éste es un programa muy sencillo, dado que apenas hay cálculo. Además, la instrucción DRAW es bastante cómoda para trazar líneas en diagonal. Los cambios a introducir en el programa para superficies tridimensionales de la lección estándar son los siguientes:

```
30 PLOT 10,Y
40 DRAW 40,0: DRAW 40,20
```

```
70 PLOT X,10
80 DRAW ),40
```

```
110 PLOT X,X/2-15
120 DRAW 0,40
```

```
150 PLOT X,50
160 DRAW 40,20
```

```
190 PLOT 2*Y-90,Y
200 DRAW 40,0
```

En la segunda parte del programa, que dibuja las caras ocultas del cubo, se harán los cambios siguiendo los mismos criterios. Las instrucciones gráficas quedarán:

```
310 PLOT 10,Y
320 DRAW 40,20: DRAW 40,0
```

```
350 PLOT X,30
360 DRAW 0,40
```

```
390 PLOT X,X/2+5
400 DRAW 0,40
```

```
430 PLOT X,10
440 DRAW 40,20
```

```
470 PLOT 2*Y - 10
480 DRAW 40,0
```

En otros programas más complejos no nos será posible utilizar la instrucción DRAW directamente y será necesario recurrir a las subrutinas 1000 y 2000.

15.2.1 Representación de funciones

El programa diseñado para representar funciones tridimensionales está perfectamente adaptado al ZX-SPECTRUM, ya que el dibujo se realiza exclusivamente a base de puntos. Únicamente hay que cambiar la línea 120 por:

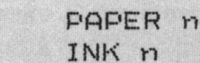
```
120 PLOT X3,-Z
```

El resto del programa queda invariable puesto que el tamaño del gráfico se ajusta a las dimensiones de la pantalla. La construcción del diagrama punto a punto supone un número enorme de cálculos. Por consiguiente, el tiempo total de ejecución será muy elevado, probablemente superior a los 15 minutos. Este tiempo depende de la función empleada y, sobre todo, de la resolución R. Ciertamente, valores menores de R conducen a gráficos de mejor calidad, pero aumentan considerablemente el tiempo de cálculo.



15.3 GRAFICOS EN COLOR

Para utilizar gráficos en color hay que asegurarse de que el televisor está perfectamente sintonizado, pues de lo contrario no se podrán apreciar con toda su claridad. Evidentemente, si trabaja con un televisor blanco y negro no apreciará los colores, sino diferentes tonos de grises. Los colores utilizados por el ZX-SPECTRUM coinciden con los expuestos en la tabla de la figura 7 del texto estándar. Los colores de fondo (del papel) y de la tinta se especifican mediante dos instrucciones distintas en lugar de utilizar la instrucción COLOR típica en otros ordenadores. La forma general de ambas instrucciones es:

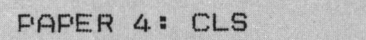


```
PAPER n
INK n
```

La primera sirve para especificar el color del fondo (PAPER significa papel) y la segunda para especificar el color del trazado (INK en español significa tinta). La letra n representa un número comprendido entre 0 y 7 que corresponde a uno de los colores de la tabla. La instrucción PAPER se encuentra escrita en la tecla C y la instrucción INK se encuentra en la tecla X.

Recordemos que para obtenerlas debemos pasar el teclado a modo E y luego pulsar la tecla manteniendo apretada SYMBOL SHIFT. Encima de las teclas de los números se encuentran escritos los nombres de los colores (en inglés) a fin de ayudarnos a recordar la asociación número-color de la tabla.

Para probar su efecto, escribimos en modo inmediato (sin número de línea).



```
PAPER 4: CLS
```

Esta instrucción ocasionará que la pantalla pase a tener color verde. Seguidamente experimentaremos con el color de la tinta. Escribamos:

```
INK 2
```

Aparentemente no ha pasado nada. Esto era de esperar, puesto que el cambio de tinta no se manifestará hasta que no escribamos algo. Tecleemos ahora

```
PRINT "LETRAS EN COLOR ROJO"
```

El mensaje de la instrucción PRINT aparecerá escrito en color rojo sobre el fondo verde de la pantalla. Esta selección de colores (verde para el papel y rojo para la tinta) permanece fija hasta que se introduzca otra instrucción PAPER o INK. El comando NEW devuelve a la pantalla sus colores iniciales, es decir, fondo blanco (color 7) con letras negras (color 0).

Como habremos observado, la instrucción PAPER solamente afecta a la zona de presentación de la pantalla. Recordemos que en el ZX-SPECTRUM, la pantalla se divide en dos zonas, una para la presentación de datos que ocupa la parte central y un borde externo, cuya parte inferior se utiliza para la entrada de datos o instrucciones. Puesto que en el borde externo no se puede escribir (excepto en la zona inferior), la instrucción PAPER no tiene efecto sobre él. Sin embargo, si queremos que absolutamente toda la pantalla adquiera un determinado color de fondo, entonces usaremos la instrucción BORDER, cuya sintaxis es:

```
BORDER n
```

La palabra BORDER significa reborde o margen. Al igual que antes, la letra n representa un número correspondiente a la tabla de colores. Esta instrucción se encuentra escrita en blanco sobre la tecla B. Probemos ahora el siguiente programa:

```
10 PAPER 1: CLS
20 BORDER 1
30 INK 6
```

Al efectuar un RUN, toda la pantalla, incluyendo reborde y zona de presentación han adquirido el color azul. En la línea 30 se ha tenido la precaución de colocar la tinta a un color claro (el amarillo), pues de lo contrario las letras negras del listado apenas se verían sobre el fondo azul oscuro. Recordemos que el comando NEW devuelve las cosas a la normalidad.

Para ver la gama de colores del ordenador utilizaremos el programa que dibuja *bandas coloreadas* que está explicado en el texto estándar. En él, cambiaremos la línea 30 por

```
30 PAPER I
```

Este programa nos dibujará 7 bandas horizontales coloreadas con un grosor de tres líneas cada una. Se ha excluido el color negro.

Para controlar el brillo, el ZX-SPECTRUM dispone de la instrucción BRIGHT, cuyo funcionamiento coincide con el descrito en el texto. La palabra BRIGHT se encuentra escrita en la tecla B. Solamente tiene dos posibilidades, 0 ó 1 según sea brillo normal o intenso. Hay que resaltar el hecho de que esta instrucción sólo afecta al brillo del color de fondo y deja invariable el color de la tinta.

En el programa que dibuja los *bandos cruzados* en la pantalla hay que introducir las siguientes variaciones:

```
 20 PAPER 1
 40 PRINT AT I,0;
 80 PAPER I/4
100 IF INT(J/6)*6<>J THEN PRINT AT J,I;" ";
```

La 20 y la 80 cambian la instrucción COLOR por PAPER. La 40 y la 100 se modifican debido a que la función AT tiene una sintaxis algo distinta en el ZX-SPECTRUM, respecto al estándar.

Utilicemos seguidamente la instrucción INK para trazar una línea de color. El programa de prueba es: (Este puede escribirlo ya.)

```
10 INK 1
20 PLOT 0,0
30 DRAW 100,100
```

Al introducir RUN, aparece una línea en diagonal de color azul. Cambiando el valor de la instrucción 10 por un 2, la recta aparecerá de color rojo y así sucesivamente con los demás colores. Es conveniente añadir la instrucción:

```
40 INK 0
```

pues de lo contrario, el listado de las instrucciones aparece también en color, dificultando su lectura.

Veamos a continuación un ejemplo de utilización conjunta del color de fondo y de la tinta. Aprovechando un fondo de color azul que incluya la zona de presentación, más el reborde de la pantalla, dibujaremos un círculo de color amarillo, de modo que parezca un astro situado en el espacio. El programa es (escríbalo):

```
10 PAPER 1: CLS
20 BORDER 1
30 INK 6
40 FOR R=2 TO 30
50   CIRCLE 150,100,R
60   NEXT R
```

Las tres últimas instrucciones forman un bucle que dibuja circunferencias concéntricas de radio creciente. Al final queda un disco de color amarillo. Sin embargo, debido a los problemas de redondeo de decimales, aparecen pequeñas imperfecciones en la superficie del disco.

Si queremos construir un disco sin defectos debemos utilizar otro sistema. En lugar de colocar líneas curvas adyacentes (en este caso circunferencias) hay que colocar segmentos adyacentes. A continuación damos el listado de esta nueva versión del programa a partir de la línea 40.

```
 40 LET P=PI/2
 50 FOR A=-P TO P STEP 0.03
 60   LET X=150+30*COS(A)
 70   LET Y=100+30*SIN(A)
 80   PLOT X,Y
 90   LET X1=150+30*COS(PI-A)
100   DRAW X1-X,0
110   NEXT A
```

Recordemos que PI es una función del ZX-SPECTRUM. Este programa dibuja segmentos superpuestos de longitud variable de modo que construyan un disco.

Queda ahora por ver el grado de precisión con que se pueden representar los gráficos a color. En el ZX-SPECTRUM, los colores se memorizan por grupos de pixels. Estos grupos son los que constituyen un carácter y forman un cuadrado de 8×8=64 pixels. El color de la tinta y del papel son únicos para todos los pixels de un mismo recuadro. El siguiente programa nos lo demuestra. Escríbalo.

```
10 INK 1
20 PLOT 0,0
30 DRAW 100,100
40 INK 4
50 PLOT 0,100
60 DRAW 100,-100
70 INK 0
```

La primera recta tiene color azul y la segunda color verde. Se observa claramente que en la instrucción el color verde invade parte de la primera recta. Esto se debe, como acabamos de explicar, a que no pueden coexistir

dos colores distintos de tinta (o de papel) en un mismo recuadro. Hay que tener en cuenta, sin embargo, que esta limitación no es muy importante y con sólo tener un poco de cuidado en el diseño de programas, podremos construir gráficos bastante sofisticados.



15.3.1 Aplicaciones

En el programa que dibuja un *diagrama de barras* hay que introducir algunas variaciones. Cambiaremos la línea 20 y la 140 del programa principal por:

```
 20 LET N=5: DIM A(N): DIM C(N)
140 INK 0
```

El primer cambio es debido a que, como sabemos, la instrucción DIM del ZX-SPECTRUM sólo admite una variable. Por el contrario, las subrutinas sufren un cambio radical. Puesto que todo el diagrama está construido de pequeños segmentos horizontales y verticales, utilizaremos directamente las instrucciones PLOT y DRAW. El listado queda:

```
300 REM Ejes
310 PLOT 24,160
320 DRAW 0,-136
330 DRAW 200,0
340 RETURN
```

```
400 REM Barra
410 INK C(I)
420 FOR J=24 TO L+24
430   PLOT P+24,J
440   DRAW 24,0
450   NEXT J
460   RETURN
```

Para *dibujar la bandera* hay que introducir algunas variaciones en el programa, unas motivadas por la función AT para posicionar el cursor, y otras motivadas por la sustitución de las instrucciones gráficas convencionales. Las instrucciones modificadas son:

```
 20 PAPER 7: CLS
 30 PAPER 2
 50 PRINT AT I,14;"          "
 80 PRINT AT I,9;
140 PRINT AT I,14;"          "
160 PAPER 7
```

Puesto que únicamente se ve afectado el color de fondo, al final del programa empleamos solamente la instrucción PAPER para que la pantalla recupere el fondo blanco. El color de la tinta se mantiene a cero (negro).

Al listado del programa utilizado para representar un prisma sólido con iluminación frontal, le añadiremos las subrutinas 1000 y 2000, ya que nos serán necesarias. Puesto que el programa principal prácticamente sólo contiene instrucciones gráficas, repetiremos el listado completo adaptado al ZX-SPECTRUM.

Programa de «Objetos sólidos»

```
 10 REM Objetos solidos
 20 FOR I=16 TO 80
 30   INK 5
 40   LET X=80: LET Y=I: GOSUB 1000
 50   LET X=104: GOSUB 2000
 60   INK 1
 70   LET X=105: GOSUB 1000
 80   LET X=112: LET Y=I+8: GOSUB 2000
 90   NEXT I
100 FOR I=80 TO 104
110   LET X=I: LET Y=80: GOSUB 1000
120   LET X=I+8: LET Y=88: GOSUB 2000
130   NEXT I
140 INK 0
```

Debe añadir las subrutinas 1000 y 2000.

Al final del programa se ha colocado la instrucción INK para que la tinta adquiera el color negro aunque, de hecho, el color azul es lo bastante oscuro para que el listado del programa destaque sobre el fondo blanco.

Para solucionar el problema de que el color de la tinta y el del papel sean demasiado parecidos y no sea posible la lectura de las instrucciones, el ZX-SPECTRUM ofrece una interesante posibilidad.

Como sabemos, los colores van de 0 a 7. Si empleamos el número 9 como código de color, entonces el ordenador lo interpreta como una indicación de que la tinta tendrá un color variable de modo que siempre ofrezca contraste con el color del papel. Para ilustrar este efecto escribimos el siguiente programa:

```
10 FOR I=0 TO 7
20   PAPER I
30   PRINT I
40   NEXT I
```

Como ya conocemos, al imprimir un carácter (un dígito en este caso) se emplea el color de la tinta para construir el símbolo y el color del papel para rellenar el resto del recuadro. En este programa el color de la tinta es

0 (negro), que es el que normalmente usa el ZX-SPECTRUM. El color del papel variará de 0 a 7. Al efectuar un RUN, aparecen 7 recuadros en vertical, cada uno de un color distinto y con un dígito escrito en negro en su interior. Lógicamente, los recuadros de color oscuro impedirán la visibilidad del dígito.

Añadiremos ahora la siguiente instrucción al programa anterior:

```
5 INK 9
```

Al efectuar un RUN, veremos que el color de la tinta es más claro o más oscuro, de forma que siempre se produzca contraste con el color del papel. El efecto de esta instrucción permanece hasta que se escribe NEW. También existe la opción equivalente para el color del papel, es decir:

```
PAPER 9
```

En este caso, es el color del papel el que debe contrastar con la tinta.

Por último, el ZX-SPECTRUM nos ofrece otra posibilidad de modificar la presentación de los resultados. Se trata de la instrucción FLASH que nos permite escribir resultados que se visualicen de forma intermitente. La palabra FLASH significa destello. Esta instrucción se encuentra en la tecla V.

La forma general es:

```
FLASH n
```

en donde n vale 0 (visualización normal) o 1 (visualización intermitente). Dado que un texto escrito de forma centelleante atrae la atención del operador, se suele utilizar esta opción cuando se escriben mensajes importantes, de forma que no pasen inadvertidos. Veamos un ejemplo: El siguiente programa convierte grados centígrados a grados Fahrenheit.

```
10 INPUT "GRADOS CENTIGRADOS:";GC
20 IF GC>-273.16 THEN GOTO 50
30   PRINT FLASH 1;"VALOR ERRONEO"
40   GOTO 10
50 PRINT "G. FAHRENHEIT=";9*GC/5+32
```

Al ejecutar el programa éste nos pregunta el valor de los grados centígrados. Si contestamos un valor inferior a −273.16 (inferior al cero absoluto) el programa nos imprime un mensaje, escrito en forma intermitente.

En el ZX-SPECTRUM, además de cambiar el estado general del color de tinta o papel, se puede modificar una de estas características mientras

se está escribiendo algo (por ejemplo mientras se está introduciendo una línea de programa). Para cambiar de modo inmediato una de estas características utilizaremos las teclas numéricas del 0 al 7 en modo E. Supongamos que estamos escribiendo la instrucción

```
10 REM
```

y queremos que el texto que sigue a continuación se escriba sobre fondo azul claro (color cyan). Para ello pasaremos el teclado a modo E y entonces pulsaremos la tecla correspondiente al 5. A continuación escribiremos el texto y se observará que las letras se escriben sobre fondo azul claro. Al final del texto volveremos al color original, repitiendo la operación con el color 7 (el blanco). También se puede cambiar el color de la tinta de modo parecido. En este caso hay que pasar el teclado a modo E y pulsar la tecla numérica (del 0 al 7) manteniendo apretadas CAPS SHIFT.

Aunque desde el punto de vista del usuario sólo se haya cambiado el color del papel o de la tinta (o de ambos), internamente el ordenador guarda dos caracteres invisibles (pero existentes), el primero de los cuales le indica si se cambia la tinta o el papel y el segundo le indica el color elegido. A causa de la existencia de estos caracteres, veremos que al utilizar la tecla DELETE para borrarlos se producen algunos efectos extraños. *En primer lugar será necesario pulsar DELETE dos veces. Después de la primera pulsación nos aparecerá un interrogante (?) o incluso es posible que aparezca alguna cosa todavía más sorprendente. Este efecto no debe preocuparnos puesto que al pulsar de nuevo DELETE desaparecerá y volveremos a la situación normal.* Por la misma razón, las teclas de desplazamiento horizontal se comportarán de forma un tanto extraña cuando el cursor atraviese una zona de cambio de color.

Además del cambio de la tinta o papel también se pueden obtener de esta forma cambios en las demás características de visualización, es decir, el brillo y el parpadeo. Para conseguirlo, pasaremos el teclado a modo E y utilizaremos las teclas 8 y 9. Las pruebas las realizaremos sobre una instrucción REM para no tener problemas adicionales con la sintaxis. Los resultados serán

8 → BRILLO NORMAL
9 → BRILLO INTENSO
CAPS SHIFT y 8 → SIN PARPADEO
CAPS SHIFT y 9 → CON PARPADEO

Sin necesidad de pasar el teclado a modo E, se puede obtener una inversión de los colores, es decir, que el color del papel pasa a ser el color de la tinta y viceversa. Esta operación se hace con CAPS SHIFT y 4 para invertir y CAPS SHIFT y 3 para volver a la situación normal. Estas teclas equivalen a la opción INVERSE ya estudiada.

Estos cambios en las características de visualización, especialmente en lo que se refiere al cambio de color, tiene gran utilidad para resaltar zonas de programa. De esta forma, en un programa que contenga varias

subrutinas, si le damos a cada una de ellas un color distinto para el listado, apreciaremos con claridad cuáles son los módulos que componen el programa completo. También se utiliza a menudo para dar más realce a las instrucciones REM.

Conviene dejar bien claro que con este procedimiento se obtiene un cambio de color (o de otra característica de visualización) en *modo inmediato.* Por el contrario, las instrucciones INK y PAPER son órdenes que afectan al color que se utilizará al efectuar un RUN, es decir, que no producirán efecto hasta que no se ejecute el programa. Haciendo una analogía, el cambio inmediato de color equivale a cambiar de lápiz mientras escribimos, y las instrucciones INK y PAPER son las especificaciones que daríamos a un impresor sobre cómo deseamos que salga impreso un texto. El único caso en el que el cambio de color en modo inmediato también tiene efecto sobre la ejecución es en los textos entre comillas (“”) que aparezcan dentro de un PRINT.

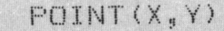


15.4 MEMORIZACION DE PANTALLA

El ZX-SPECTRUM dispone de tres instrucciones para consulta y almacenamiento del estado de una pantalla. Estas instrucciones son POINT, ATTR y SCREEN$. A continuación describiremos el funcionamiento de cada una de ellas.

15.4.1 Instrucción POINT

Esta instrucción se utiliza para averiguar el estado de activación de un punto (o pixel) de la pantalla. Por tanto, sustituirá a la instrucción convencional ESTADO definida en el texto general. La palabra POINT significa punto. En realidad, POINT más que una instrucción es una función cuya sintaxis es:



Esta función tiene dos argumentos, X e Y, que corresponden a las coordenadas del pixel cuyo estado se quiere determinar. A diferencia de lo que es costumbre en este ordenador, en este caso los paréntesis son necesarios y obligatorios. La palabra POINT se encuentra escrita en la tecla del 8.

Esta función sólo puede tener dos resultados: 1 si el pixel está activado o 0 si está desactivado. Recordemos que un punto de la pantalla está activado si tiene el color de la tinta y desactivado si tiene el color de fondo o de papel, sean cuales sean el color de tinta y papel.

Veamos algunos ejemplos de funcionamiento en modo inmediato. Escribamos:

```
CLS: PRINT POINT(10,10)
```

Obtendremos un cero como resultado, puesto que toda la pantalla está en blanco. Tecleemos seguidamente:

```
PLOT 10,10
```

como era de esperar, aparecerá un punto de color negro en las coordenadas (10,10). Si repetimos la orden:

```
PRINT POINT(10,10)
```

entonces comprobamos que el resultado es 1. Insistimos en que el color de la tinta no afecta al estado de activación. Por ejemplo, demos la siguiente orden:

```
INK 7: PLOT 50,50: INK 0
```

En esta orden, colocamos la tinta a color blanco, dibujamos un punto en (50,50) que no será visible puesto que coincide con el color del papel y ponemos la tinta de nuevo a color negro. Escribamos ahora

```
PRINT POINT(50,50)
```

y el resultado será 1, puesto que el pixel está activado aunque sea de modo invisible. Por el contrario, si tecleamos

```
PRINT POINT(50,49)
```

que corresponde al pixel adyacente, el resultado es cero ya que no ha sido activado.

En el programa que dibuja un punto móvil en la pantalla, introduciremos varias modificaciones, entre ellas el cambio de la función ESTADO por POINT. El listado queda:

Programa de «Punto móvil»

```
 10 REM Punto movil
 20 INPUT "POSICION:";P
 30 PLOT P,1
 40 DRAW 0,100
 50 PLOT 1,50
 60 FOR X=2 TO 200
 70   IF POINT(X,50)=1 THEN GOTO 110
 80   PLOT INVERSE 1,X-1,50
 90   PLOT X,50
100   NEXT X
110 LET M=X-2
120 FOR X=M TO 1 STEP -1
130   PLOT INVERSE 1,X+1,50
140   PLOT X,50
150   NEXT X
```

En este programa se ha preferido utilizar directamente PLOT y PLOT INVERSE en lugar de recurrir a las subrutinas de dibujo, a causa de que sólo se trazan puntos aislados. En la línea 70 se ha sustituido la función convencional ESTADO por POINT. En la línea 60 podemos aprovechar al máximo la pantalla cambiando el 200 por 255. El estado de activación o desactivación es la única característica que se memoriza para cada pixel por separado. Las demás características de visualización, llamadas también *atributos,* se memorizan únicamente para grupos de bits. Este grupo de bits es el recuadro o posición de escritura de un carácter que, como ya se ha mencionado anteriormente, tiene unas dimensiones de 8×8=64 pixels. Un cambio de atributos en uno de los pixels repercute inmediatamente en todos los demás del grupo. La única característica que se memoriza de forma independiente para cada uno es la activación o desactivación.

Los atributos son: el color del papel, el color de la tinta, el brillo y el parpadeo (FLASH). Para almacenar toda esta información sólo se precisa un byte. Dado que nuestra pantalla tiene 24 líneas por 32 columnas, se tiene un total de 768 bytes de la memoria central para almacenar todos los posibles atributos de la pantalla completa. La información de los atributos de un carácter está distribuida entre los 8 bits del byte de la forma que se muestra en la figura 1. El bit situado a la izquierda contiene la información del parpadeo o FLASH. Si vale 1, la posición es destelleante. Si vale 0, la posición es fija. El siguiente bit contiene la información del brillo. Si vale cero, la posición tiene brillo normal y si vale 1 tiene brillo intenso. Para almacenar los colores necesitaremos más de un bit. Como recordamos de la tabla de códigos de color, se utilizan 3 bits para representar un color. Entonces, los 6 bits que quedan se reparten en 3 bits para el color del papel y 3 bits para el color de la tinta.

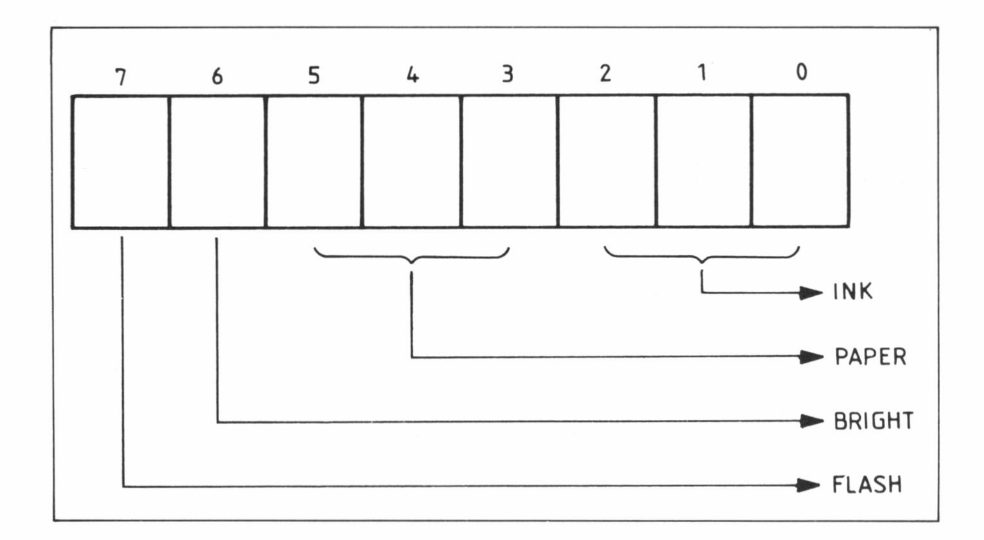


Figura 1: Correspondencia entre bits y atributos de una posición de pantalla.

El ZX-SPECTRUM dispone de una función para consultar el estado de los atributos de una posición de pantalla. Antes de seguir con el texto tal vez sea conveniente dar un repaso al sistema binario explicado en la unidad 2, especialmente por lo que se refiere a la conversión decimal-binario. La función se denomina ATTR y su sintaxis es

```
ATTR(Y,X)
```

La palabra ATTR proviene de la abreviatura de «attribute» (atributo en español), y se encuentra escrita en la letra L. Esta función nos devuelve el estado de los atributos de una posición de pantalla. No olvidemos que en esta función los argumentos *X* e *Y* se refieren a la *posición de carácter* en la pantalla. Por tanto *X* estará comprendido entre 0 y 31. A su vez, *Y* estará comprendido entre 0 y 21. No hay que confundir con la función POINT cuyos argumentos son las coordenadas de un pixel, y por tanto sus argumentos están comprendidos entre 0 y 255, y entre 0 y 175 respectivamente.

El resultado es un número comprendido entre 0 y 255. Este margen de valores era de esperar puesto que la función nos devuelve el valor de 1 byte y ya sabemos que con 8 bits tenemos un total de 256 combinaciones distintas, que van desde cero (los 8 bits a cero) hasta 255 (los 8 bits a 1). Ahora bien, de este número debemos extraer la información correspondiente a cada atributo.

La aportación de un bit al valor global se obtiene multiplicando por 2 tantas veces como indique su posición. Por ejemplo, el bit séptimo, si vale 1 aportará 128 al valor total (2 elevado a 7).

De esta forma, si el resultado de ATTR es mayor o igual que 128 significa que el bit séptimo vale 1 y, por consiguiente, la posición es destelleante.

A continuación veremos un programa que determina el estado de todos los atributos de una posición de la pantalla. En primer lugar escribe 20 posiciones con las características de visualización elegidas al azar. Segui-

damente, mediante la función ATTR determina y escribe los atributos de una posición determinada. El listado es:

Programa de «determinación de atributos»

```
 10 REM Determinacion atributos
 20 FOR J=1 TO 20
 30   LET F=INT(RND+1): LET B=INT(RND+1)
 40   LET P=INT(RND*8): LET I=INT(RND*8)
 50   LET X=INT(RND*32): LET Y=INT(RND*22)
 60   FLASH F: BRIGHT B
 70   PAPER P: INK I
 80   PRINT AT Y,X;"*";
 90   NEXT J
100 INK 0: PAPER 7: FLASH 0: BRIGHT 0
110 INPUT "COLUMNA:";X
120 INPUT "FILA:";Y
130 LET A=ATTR(Y,X)
140 IF A<128 THEN PRINT "SIN DESTELLO": GOTO 170
150   PRINT "CON DESTELLO"
160   LET A=A-128
170 IF A<64 THEN PRINT "BRILLO NORMAL": GOTO 200
180   PRINT "BRILLO INTENSO"
190   LET A=A-64
200 LET COLORP=INT(A/8)
210 LET COLORT=A-8*COLORP
220 PRINT "PAPEL:";COLORP
230 PRINT "TINTA:";COLORT
```

La primera parte del programa formada por el bucle, que va desde la línea 20 hasta la 90, imprime 20 asteriscos con atributos elegidos al azar. La opción de FLASH, el brillo, el color de fondo y el color de la tinta se calculan con la función RND y se almacenan en las variables F, B, P e I respectivamente. La posición de la pantalla se determina también al azar. Cuando se han impreso todos ellos, volvemos a los colores y atributos normales. Entonces el programa pide al usuario que elija una posición. Mediante la función ATTR, almacenamos en la variable A el valor del byte de los atributos.

A continuación, el programa comprueba si el bit 7 vale 1 o 0, en otras palabras, si A es mayor o igual que 128. Si no es así, significa que la opción de destello esta inhibida. De lo contrario, la opción está activada. Entonces hay que restar 128 a esta variable de modo que este bit no intervenga en las siguientes comparaciones.

El mismo proceso se sigue para determinar el brillo de la posición. Para los colores el procedimiento es un poco distinto, puesto que manejamos tres bits a la vez.

Al ejecutar este programa es posible que al escribir los resultados se borre la posición cuyos atributos se estaban averiguando por coincidir ambos en la misma zona de pantalla. Para el programa no constituye ningún inconveniente puesto que todos los atributos se habían memorizado en la variable A.

Seguramente encontramos que es algo difícil localizar la posición en la pantalla. Por ello introduciremos una modificación en el programa, de tal modo que nos numere las filas y las columnas. Las instrucciones a añadir son:

```
300 PRINT AT 0,0;
310 FOR I=1 TO 3
320   PRINT "0123456789";
330   NEXT I
340 PRINT AT 0,0;
350 FOR I=0 TO 21: PRINT I: NEXT I
360 RETURN
```

Además, en el programa inicial añadiremos las líneas:

```
105 GOSUB 300
240 STOP
```

Al ejecutar el programa veremos que nos escribe una numeración en los márgenes de la pantalla, facilitando así la localización de un carácter determinado.

15.4.2 Memorización global

Nos queda por último la operación de almacenamiento de una pantalla completa. Para ello utilizaremos la instrucción SCREEN$ en combinación con la instrucción SAVE. La palabra SCREEN significa pantalla. En el ZX-SPECTRUM la memorización se realiza sobre una cinta magnética y se almacena como si fuera una tabla de datos. La memorización es siempre global, aunque el dibujo ocupe sólo una pequeña parte de la pantalla. La utilización primordial de esta instrucción será para almacenar fondos de pantalla o dibujos complejos que requieren un tiempo elevado de cálculo y, a la vez, un gran número de instrucciones para construirlo. De esta forma, el dibujo se puede elaborar mediante un programa previo y luego almacenar el resultado. Por tanto nos ahorraremos todas estas instrucciones del programa que haga uso del dibujo, ya que será suficiente colocar una sola instrucción para recuperar el dibujo de la cinta.

La instrucción SCREEN$ se encuentra escrita en la tecla K. Para realizar la grabación escribiremos:

```
SAVE "nombre" SCREEN$
```

en donde «nombre» es el nombre del archivo. Cualquier otro nombre será también válido, siempre que no supere las diez letras.

Para obtener en pantalla el dibujo almacenado, escribiremos:

```
LOAD "nombre" SCREEN$
```

El nombre del archivo de la instrucción LOAD debe coincidir lógicamente con el utilizado para SAVE.

En todo caso sería conveniente repasar el capítulo 8, en el segundo tomo, que trata precisamente de cómo se maneja una cinta. En lugar de la palabra SCREEN$ se puede utilizar CODE 16384,6912, es decir:

```
SAVE "prueba" CODE 16384,6912
LOAD "prueba" CODE 16384,6912
```

La función CODE es la misma que hemos empleado para obtener los códigos ASCII de los símbolos del teclado, aunque aquí tiene un significado distinto. Ambas formas de escribir las instrucciones son totalmente equivalentes. El número 16384 es la posición de memoria (en la memoria central) donde empieza el almacenamiento de la información de la pantalla y el segundo, 6912 es el número de bytes que se almacenan.

Para poder realizar prácticas con esta instrucción necesitamos que exista previamente un dibujo en la pantalla. Por tanto, elaboraremos un programa que construya un dibujo bastante completo, a fin de que se aprecie claramente la ventaja de almacenarlo en lugar de recalcularlo de nuevo cada vez. El programa nos dibuja un escenario con cierto aspecto de ciencia ficción, que tal vez nos sea útil para un futuro programa de juegos.

El listado del programa es el siguiente:

```
 10 REM Escenario
 20 PAPER 1: CLS
 30 PAPER 4: PRINT AT 12,0;
 40 FOR I=1 TO 320
 50   PRINT " ";
 60   NEXT I
 70 INK 6
 80 FOR I=0 TO 7
 90   PLOT 32*I,0
100   DRAW 32-8*I,80
110   READ A
120   PLOT 0,A: DRAW 255,0
130   NEXT I
140 PLOT 0,65: DRAW 8,15
150 PLOT 255,2: DRAW -31,78
160 PLOT 255,66: DRAW -8,14
170 FOR A=-PI/2 TO PI/2 STEP 0.03
180   LET X=210+30*COS(A)
```

```
190     LET Y=140+30*SIN(A)
200     PLOT X,Y
210     LET X1=210+30*COS(PI-A)
220     DRAW X1-X,0
230     NEXT A
240 PAPER 0
250 FOR I=10 TO 14
260     PRINT AT I,7;"     ";
270     NEXT I
280 INK 7
290 FOR I=55 TO 95
300     PLOT 80,I: DRAW 8,8
310     NEXT I
320 FOR I=56 TO 79
330     PLOT I,97: DRAW 8,8
340     NEXT I
350 INK 9: PAPER 7
360 DATA 0,25,45,60,70,75,78,80
```

La línea 20 pasa la pantalla a color azul. Las siguientes instrucciones, de la 30 a la 60, construyen una zona de color verde en la mitad inferior de la pantalla. Este cambio de color se realiza a base de la impresión repetida de espacios en blanco. Esta zona de color verde debe representar una superficie plana. Para dar sensación de profundidad utilizaremos el procedimiento de retícula combinado con la perspectiva. De esta operación se encarga el bucle que va de la línea 80 a la 130, el cual construye un enrejado cuyas líneas se van haciendo más próximas a medida que nos acercamos al centro de la pantalla. Las instrucciones 140, 150 y 160 dibujan los segmentos que faltan para completar la retícula. Observemos que para trazar las líneas horizontales se emplea una instrucción READ (línea 110) que lee los datos de la instrucción 360. Se ha preferido este sistema puesto que los cálculos eran demasiado complicados.

A continuación, el programa dibuja una figura en forma de Sol en la parte superior derecha de la pantalla. Para ello seguiremos el método indicado anteriormente para trazar discos coloreados. El bucle formado por las líneas comprendidas entre la 170 y la 230 realiza este cometido.

Finalmente, dibujaremos un prisma de sección cuadrada sobre la parte izquierda de la superficie. Utilizaremos el color negro para las superficies no iluminadas y el color blanco para las iluminadas. En este caso hay que tener en cuenta que la iluminación no es frontal sino lateral (procedente del disco anterior). Por tanto, la cara frontal será de color negro.

Una vez realizado el dibujo en pantalla, lo grabaremos en la cinta. Escribiremos (sin pulsar ENTER):

```
SAVE "escenario" SCREEN$
```

Pondremos en marcha la cinta y a continuación pulsaremos ENTER. El ordenador graba los datos siguiendo el proceso habitual. Conviene remarcar que se graba el contenido de la pantalla, pero no el programa. Si se desean grabar las instrucciones hay que utilizar el comando SAVE de la forma normal.

Seguidamente borraremos la pantalla mediante la instrucción CLS. Es mejor, de momento, no borrar el programa, puesto que la grabadora requerirá probablemente algunos ajustes antes de que funcione correctamente. Una vez tenemos la pantalla borrada, rebobinamos el cassette, lo detenemos y escribimos (sin pulsar ENTER):

```
LOAD "escenario" SCREEN$
```

Ponemos en marcha la grabadora e inmediatamente pulsamos ENTER. Si la grabación ha tenido éxito, veremos como reaparece en pantalla el dibujo anterior. Si no ha tenido éxito, repetiremos las operaciones ajustando los controles de volumen y de tono de la grabadora.

Puede que nos parezca un poco extraña la forma en que se reconstruye en pantalla el dibujo memorizado. Ello es debido a la equivalencia interna entre los puntos de la pantalla y las posiciones de memoria del ordenador. En el ZX-SPECTRUM, esta equivalencia se hace a base de líneas múltiples de 8.

Si hemos tenido éxito en la grabación y posterior recuperación del dibujo, ya podemos borrar el programa anterior. El dibujo podrá ser reproducido en la pantalla en cualquier momento. Además, la instrucción LOAD puede formar parte de un nuevo programa. De esta forma, con una sola instrucción, el programa reproduce todo el dibujo anterior para el cual se han necesitado originalmente más de treinta líneas.

Por ejemplo, el programa:

```
10 LOAD "escenario" SCREEN$
20 PRINT AT 1,5;"PRUEBA SCREEN$"
```

Superpondrá el texto «PRUEBA SCREEN$» sobre el dibujo. Como es lógico, al escribir RUN habrá que pone en marcha la grabadora, de modo que la instrucción 10 sea capaz de encontrar la información grabada.



15.5 SONIDO

El ordenador tiene la capacidad de producir sonido. En este sentido, vamos a ver dos maneras de aprovechar estas posibilidades.

La primera consiste en utilizarlo para «reproducir» sonidos que se hayan introducido previamente en el programa, de manera que al ejecutarlo, obtendremos la música que hayamos «escrito».

A continuación presentamos un programa para ver cómo se realiza en forma práctica.

Programa de «sonido»

```
NEW
  10 CLS
  20 REM Presentacion de pantalla
  30    BORDER 7 : PAPER 5 : INK 1
  40    PRINT "Cancion de cuna"
  50    PRINT TAB(10);"Brahms"
  60 REM Lectura y ejecucion de las notas
  70    READ N
  80    FOR I = 1 TO N
  90       READ A
 100       BEEP 0.6,A
 110       NEXT I
 120    PAUSE 100
 130    RESTORE
 140    GOTO 70
1000 REM Almacen de los valores de las notas
1010 DATA 27 :      REM Numero de notas de la cancion
1020 DATA 9,9,12,9,9,12,9,12,17,16,14,14,12
1030 DATA 7,9,10,7,7,9,10,7,10,16,14,12,16,17
```

El programa tiene varias partes. En primer lugar, hay una parte de presentación de pantalla, que escribe el título y el autor de la canción (líneas 10 a 50).

A continuación se inicia la lectura del almacén. El primer valor que se lee corresponde al número de valores del almacén, es decir, el número de notas de la canción. Este número indica por tanto cuántos datos hay que leer. Para ello, escribiremos un bucle (líneas 80 a 110), en el cual se leerá el valor correspondiente a cada nota (línea 90) y se producirá a continuación el sonido correspondiente a la misma (línea 100), mediante la instrucción BEEP. Recuerde el funcionamiento de esta instrucción. El primer número indica la duración del sonido; el segundo indica qué sonido se ha de producir. En la figura 2 se muestra la equivalencia entre las notas musicales y los números que les corresponden.

En este programa, tal como lo hemos escrito, la duración del sonido es la misma para cada nota, ya que el primer número escrito en la instrucción BEEP es una constante.

El segundo valor lo vamos leyendo del almacén (líneas 1000 a 1030), que —tal como ya hemos visto— contiene en primer lugar el número de notas (línea 1010) y a continuación los valores correspondientes a ellas (líneas 1020 y 1030).

Una vez finalizado el bucle la canción ha terminado. Hacemos entonces una pequeña pausa (línea 120), reinicializamos el indicador de posición del almacén (línea 130) y volvemos de nuevo al principio de la canción. De

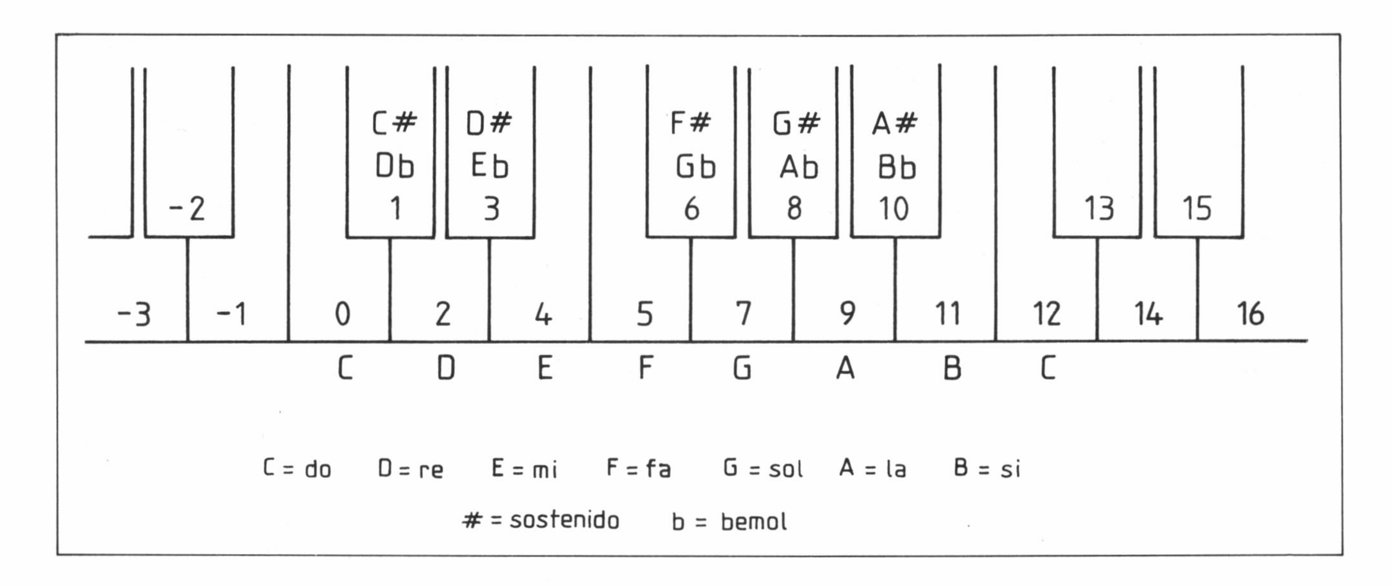


Figura 2: Correspondencia entre las notas musicales y los números equivalentes para la instrucción BEEP.

esta forma el programa se irá ejecutando indefinidamente hasta que lo detengamos pulsando la tecla de interrupción

Tal como hemos dicho, la duración es la misma para todas las notas; la música no acaba de sonar bien. Si conocemos cuál ha de ser la duración correcta de cada nota, podemos almacenar el valor correspondiente junto al valor del sonido. De esta forma leeremos cada vez dos valores, en primer lugar la duración y en segundo lugar el tono, para cada nota. El programa quedará ahora:

Programa de «Sonido y duración de las notas»

```
 NEW
  10 CLS
  20 REM Presentacion de pantalla
  30    BORDER 7 : PAPER 5 : INK 1
  40    PRINT "Cancion de cuna"
  50    PRINT TAB(10);"Brahms"
  60 REM Lectura y ejecucion de las notas
  70    READ N
  80    FOR I = 1 TO N
  90       READ A,B
 100       BEEP A,B
 110       NEXT I
 120    PAUSE 100
 130    RESTORE
 140    GOTO 70
1000 REM Almacen
1010 DATA 27 :       REM Numero de notas de la cancion
1020 DATA .4,9,.4,9,.5,12,.4,9,.4,9,.5,12
1025 DATA .3,9,.3,12,.3,17,.4,16,.4,14,.4,14,.3,12
1030 DATA .3,7,.3,9,.4,10,.3,7,.3,7,.3,9,.4,10
1040 DATA .3,7,.3,10,.4,16,.3,14,.4,12,.4,16,.5,17
```

De esta forma, usted podrá introducir la música que desee modificando únicamente el contenido del almacén, y naturalmente la presentación de pantalla.

Ya sabemos cómo utilizar el ordenador para reproducir sonido «pregrabado». Vamos a ver ahora otra posibilidad: utilizarlo como un instrumento. Vamos a escribir un programa que nos permita utilizar el ZX-SPECTRUM de forma semejante a un piano. Asignaremos a cada tecla una nota, de forma que al ejecutar el programa, pulsando la tecla, obtengamos el sonido correspondiente a la misma. Usted podrá definir tantos sonidos distintos —tantas notas— como teclas tiene el ordenador.

A continuación veremos el programa que nos permitirá hacerlo. Hemos escogido arbitrariamente las teclas situadas en la mitad del teclado (fila de la letra Q y fila de la letra A). Las de la línea inferior corresponderían a las teclas blancas del piano, las de la línea superior a las teclas negras, tal como se indica en la figura 3.

Programa de «ordenador como piano»

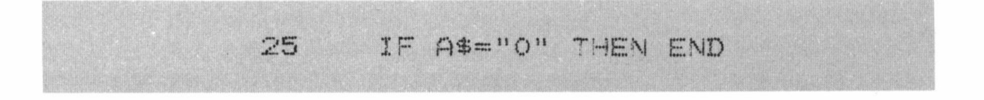
```
NEW
10 REM Utilizacion del ordenador como instrumento
15  LET D = 0.4
20  LET A$=INKEY$ : IF A$="" THEN GOTO 20
30  IF A$="A" THEN BEEP D,0 : GO TO 20 : REM DO
40  IF A$="W" THEN BEEP D,1 : GO TO 20 : REM DO#
50  IF A$="S" THEN BEEP D,2 : GO TO 20 : REM RE
60  IF A$="E" THEN BEEP D,3 : GO TO 20 : REM RE#
70  IF A$="D" THEN BEEP D,4 : GO TO 20 : REM MI
80  IF A$="F" THEN BEEP D,5 : GO TO 20 : REM FA
90  IF A$="T" THEN BEEP D,6 : GO TO 20 : REM FA#
100 IF A$="G" THEN BEEP D,7 : GO TO 20 : REM SOL
110 IF A$="Y" THEN BEEP D,8 : GO TO 20 : REM SOL#
120 IF A$="H" THEN BEEP D,9 : GO TO 20 : REM LA
130 IF A$="U" THEN BEEP D,10 : GO TO 20 : REM LA#
140 IF A$="J" THEN BEEP D,11 : GO TO 20 : REM SI
150 IF A$="K" THEN BEEP D,12 : GO TO 20 : REM DO
160 GOTO 20
```

Veamos el funcionamiento del programa. Para determinar qué tecla se pulsa se utiliza la función INKEY$. La tecla pulsada se asigna sobre la variable A$. Mientras no se pulse ninguna tecla, la variable A$ es nula, por tanto el programa queda en un bucle (línea 20), del que no sale hasta que no pulsemos una tecla. Cuando se pulsa una tecla, la variable A$ toma el valor de la misma (será por tanto distinta del texto nulo) y pasamos a la línea siguiente, donde se empieza a comparar la tecla pulsada con las que se han definido previamente (líneas 30 a 150). Cuando coincide la tecla pulsada con alguna de las predefinidas, se produce el sonido que se ha fijado previamente para ella, y se vuelve a la línea 20, donde el programa queda a la espera de que se pulse otra tecla. Junto a cada línea se ha escrito la nota a la que corresponde en la escala musical. Si usted no posee conocimientos de música no importa, prescindiendo de las notas podrá igualmente improvisar la música que desee.

Figura 3: Correspondencia entre las teclas del piano y las utilizadas en el teclado del ordenador.

La duración de cada nota se ha fijado a un valor determinado, idéntico en todos los casos. Sin embargo, hemos utilizado una variable para indicarlo, porque si usted desea aumentar o disminuir la duración de cada nota bastará modificar el contenido de esta variable. En este sentido, no podemos variar la duración de cada nota a voluntad, tal como haríamos en un instrumento musical; no queda otra posibilidad que fijarlo previamente; siempre que pulsamos la misma tecla, obtendremos una nota de la misma duración.

Para detener el funcionamiento del programa deberemos pulsar la tecla de interrupción, o bien añadir la línea:



```
25 IF A$="0" THEN END
```

con lo cual detendremos la ejecución pulsando la tecla del 0.

El programa quedará ahora:

```
 10 REM Utilizacion del ordenador como instrumento
 15   LET D=0.4
 20   LET A$=INKEY$ : IF A$="" THEN GOTO 20
 25   IF A$="O" THEN END
 30   IF A$="A" THEN BEEP D,0 : GO TO 20 : REM DO
 40   IF A$="W" THEN BEEP D,1 : GO TO 20 : REM DO#
 50   IF A$="S" THEN BEEP D,2 : GO TO 20 : REM RE
 60   IF A$="E" THEN BEEP D,3 : GO TO 20 : REM RE#
 70   IF A$="D" THEN BEEP D,4 : GO TO 20 : REM MI
 80   IF A$="F" THEN BEEP D,5 : GO TO 20 : REM FA
 90   IF A$="T" THEN BEEP D,6 : GO TO 20 : REM FA#
100   IF A$="G" THEN BEEP D,7 : GO TO 20 : REM SOL
110   IF A$="Y" THEN BEEP D,8 : GO TO 20 : REM SOL#
120   IF A$="H" THEN BEEP D,9 : GO TO 20 : REM LA
130   IF A$="U" THEN BEEP D,10 : GO TO 20 : REM LA#
140   IF A$="J" THEN BEEP D,11 : GO TO 20 : REM SI
150   IF A$="K" THEN BEEP D,12 : GO TO 20 : REM DO
160   GOTO 20
```

Tal como hemos dicho, la elección de las teclas es arbitraria. Visto ya el mecanismo del programa, usted puede realizar en él las modificaciones que desee, tanto para añadir nuevas teclas, como para cambiar las que se han escogido.



Indice

Capítulo 14

Capítulo 15



SPECTRUM

4

- La programación estructurada
- El concepto de bloque
- El concepto de refinamiento
- Estructura secuencial
- Estructura condicional
- Alternativas simple y múltiples
- La instrucción ON... GOTO... y ON... GOSOB...
- Estructura de repetición e iterativa
- El bucle MIENTRAS
- El bucle HASTA
- Bucles generalizados
- La programación estructurada. Ordinogramas
- Caracteres gráficos
- Gráficos de alta resolución
- Gráficos bidimensionales
- Dibujos geométricos
- Diagramas de barras, circulares y X-Y
- Superposición de líneas
- Operaciones bidimensionales
- Dibujos tridimensionales
- Gráficos en color
- Memorización de pantalla
- Producción de sonido
- Diseño de una calculadora
- Aprovechamiento del editor de tabla

ceac